MASTER IN COMPUTER SCIENCE

Generalization/specialization abstraction structures
Theoretical study and integration in a database design workbench

Zeippen, Jean-Marc

Award date:
1990

Awarding institution:
University of Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Generalization/specialization abstraction structures

-

Theoretical study and integration in a database design workbench

Jean-Marc ZEIPPEN

Mémoire présenté sous la direction du
Professeur Jean-Luc HAINAUT
pour l'obtention du titre de
Licencié et Maître en Informatique

Année académique 1989-1990

Abstract

Abstract

The generalization/specialization abstraction structures offer a powerful specification tool. This work first attempts to identify the different facets of the concept, as far as conceptual database design is concerned: indeed, generalization/specialization structures are part of numerous (semantic) data models but there are many variations around the definitions and the ways in which they are used. From this theoretical analysis, we define a useful and consistent subset of these structures in order to include them in an existing database design workbench based on the entity-relationship approach. The proposed construct, called the category, is introduced in the workbench model; the existing processors are modified, while new functions are proposed (a special care is given to transformations).

Keywords: abstraction mechanism, CASE, conceptual modelling, database design, database design workbench, entity-relationship approach, generalization/specialization, semantic data model, transformation

Résumé

Les structures d'abstraction de généralisation/spécialisation offrent un puissant outil de spécification. Ce travail essaie d'abord d'identifier les différentes facettes de ce concept, au niveau de la construction conceptuelle d'une base de données : en effet, les structures de généralisation/spécialisation font partie de nombreux modèles de données (sémantiques) mais il y a des variations à propos des définitions et de leurs usages. A partir de cette analyse théorique, nous définissons un sous-ensemble utile et cohérent de ces structures, de façon à les inclure dans un atelier de conception de bases de données existant basé sur l'approche entité-association. La construction proposée, appelée catégorie, est introduite dans le modèle de données de l'atelier; les processeurs existant sont modifiés, tandis que de nouvelles fonctions sont proposées (les transformations sont particulièrement étudiées).

Mots-clés: AGL, approche entité-association, atelier de conception de bases de données, conception de base de données, généralisation/spécialisation, mécanisme d'abstraction, modèle de données sémantique, modélisation conceptuelle, transformation

Acknowledgments

I am particularly grateful to Professor Jean-Luc HAINAUT, promoter of this dissertation, for the trust he put in me, for the freedom he let me, and last but not least, for the guidance and the advice he gave me. I take this opportunity to offer him my deepest thanks.

I also thank his collaborators, Mario CADELLI and Bernard DECUYPER, for their explanations on the TRAMIS database design workbench.

I would also like to thank all the people who welcomed and helped me during my training (granted by the European COMETT program) in the VIA group at Digital Equipment Corporation in Valbonne (France). I learned a lot while working on the DECdesign CASE tool and when teaching a database design course; the experience I gained there helped me considerably in the realization of this work. A special thank to Philippe DOLS, my adviser during these months, for the technical abilities he gave me the opportunity to learn.

I owe a great debt of thanks to my parents for caring for my education, and still giving me full support during my university studies. I want to thank my brother, my sister (who typed most of these pages), and the other members of my family for being by my side when I needed.

I cannot forget the schoolteachers of my village and the teachers of the Institut Sainte-Marie of Arlon who helped me to broaden my mind. I thank more especially Mr. Paul JEAN for his enthusiasm in English teaching, and for correcting the English of earlier drafts. I also acknowledge the academic staff of the Facultés Universitaires Notre-Dame de la Paix of Namur for both the quality of their lectures and the pedagogical aspects of university education.

Finally, I want to thank all the other people who have more or less contributed to this work.

« ▪ *What kind of world are we creating with computers?*

▪ *What kind of world do you want to live in?*

▪ *How can you use computers, amongst other things, to create that world?*

▪ *Are you doing those things now? Or, at least, is your work consistent with your desired world view?*

▪ *Whatever you are doing now with computers, you are changing your world. Is it for the better?* »

(M. L. Brodie - 1988)

Contents

Introduction 1
- Background 1
- Objective and motivation 2
- Related works 2
- Main ideas 3
- Outline 3
- Note to the reader 4

Chapter 1
The generalization/specialization abstraction mechanism 5

1.1. Abstraction mechanisms 5
- 1.1.1. Four main abstraction mechanisms 6
  - A. Classification 6
  - B. Aggregation 6
  - C. Association 7
  - D. Generalization 7
- 1.1.2. Hierarchies of abstractions 8

1.2. The generalization/specialization abstraction mechanism in software development lifecycle 9
- 1.2.1. Software development lifecycle 9
- 1.2.2. Software development based on generalization/specialization 9

1.3. Generalization/specialization in programming, artificial intelligence, and database areas 10
- 1.3.1. Generalization/specialization and object-oriented programming 11
  - A. Object orientation 11
  - B. Generalization/specialization and objects 12
- 1.3.2. Generalization/specialization and artificial intelligence 12
  - A. Knowledge representation and manipulation 12
  - B. Generalization/specialization in knowledge representation and reasoning 13
- 1.3.3. Generalization/specialization and databases 13
  - A. Database models 13
  - B. Generalization/specialization constructs in database models 15

Chapter 2
Database design 18

2.1. Database design methodology 18
- 2.1.1. The information system and the database 18
- 2.1.2. Database design methodology 18
  - A. Design steps 19
  - B. Data models 19
  - C. Computer-aided design 20

2.2. Conceptual modelling 21
- 2.2.1. The UoD and the database 21
  - A. The UoD 21

B. The database 21
2.2.2. The conceptualization or conceptual modelling process 22
2.2.3. Conceptual models 23
2.2.4. Building a conceptual schema 23

Chapter 3
Generalization/specialization structures in database models 25

3.1. The generalization/specialization basic block: the *is-a* relation 26
3.1.1. The *is-a* relation 26
A. Definitions 26
B. Example 28
C. Discussion 28
3.1.2. The *is-a* graph 28
A. Properties of the *is-a* relation 28
B. Definitions 29
C. Properties of the *is-a* graph 29
D. Discussion 30

3.2. Inheritance 31
3.2.1. The inheritance inference rule 31
3.2.2. Downward and upward inheritance 32
A. Definitions 32
B. Example 33
C. Discussion 33
3.2.3. Redefinition and inhibition constraints 34
A. Definitions 34
B. Discussion 35
3.2.4. Multiple inheritance 35
3.2.5. Specialization relationship inheritance 35

3.3. Class constraints 36
3.3.1. Disjunction constraint 36
A. Definition 36
B. Example 36
C. Discussion 36
3.3.2. Covering constraint 37
A. Definition 37
B. Example 37
C. Discussion 37
D. Definition 38
E. Example 38
3.3.3. Partition constraint 38
A. Definition 38
B. Example 38

3.4. The generalization/specialization criterion 39
3.4.1. Semantic criterion 39
3.4.2. Database-defined specialization 39
A. Definition 39
B. Example 40
C. Discussion 40
3.4.3. User-defined specialization 41
A. Definition 41
B. Example 41
C. Discussion 41

3.5. *Is-a* constructs 41
3.5.1. The problem of defining a generalization/specialization construct 42
3.5.2. Different proposals 43
A. The cluster of Smith and Smith 43
B. The category of Sakai 43
C. Three types of generalization/specialization constructs 43
D. The cluster of Davis and Bonnel 43
E. The subset hierarchy and the generalization hierarchy 44
F. The subset hierarchy and the exclusive (complete) generalization hierarchy 44
G. The *is-a* interconnection in SDM 44

3.6. Another generalization/specialization relation: the *may-be-a* relation 44
3.6.1. The *may-be-a* relation 44
3.6.2. Class constraints 45
3.6.3. The *may-be-a* constructs 46
A. The alternative generalization 46
B. The category 46
C. Simple generalization, alternative generalization, multiple generalization, selective generalization 46

3.7. Generalization/specialization for relationship types and attributes 47

Chapter 4
Generalization/specialization and database design activities 48

4.1. Introduction of generalization/specialization constructs in schemata 48
4.1.1. Intersection between entity domains 49
A. Identical entity domains 49
B. Inclusion of a domain into another domain 50
C. Two or more domains are overlapping 50
D. Two or more disjoint domains 50
4.1.2. Descriptive considerations 50
4.1.3. Global considerations 50

4.2. Transformations concerning generalization/specialization constructs 51
4.2.1. Elementary transformations 52
A. Representation of *is-a* relations by relationship types 52
B. Representation of the sole generic entity type 53
C. Representation of the sole specific entity types 54
4.2.2. Complex transformations 55

Chapter 5
The TRAMIS database design workbench 56

5.1. TRAMIS architecture 56
5.1.1. User level explanations 56

5.1.2. Technical level explanations 57

5.2. TRAMIS model 58
5.2.1. User level explanations 58
A. Schema 58
B. Entity type 58
C. Relationship type 59
D. Attribute 60
E. Identifier group 61
F. Textual descriptions 61
G. Technical note 61
H. Origin 61
I. Statistical aspects 62
5.2.2. Technical level explanations 63
A. The E-R schema of the specification database 63
B. The GAM schema of the specification database 65

5.3. TRAMIS processors 65
5.3.1. User level explanations 65
A. Management 66
B. Consultation 66
C. Transformation 66
D. Model compliance checking 68
E. Production of executable descriptions 69
F. Reporting 69
G. Import/export 69
5.3.2. Technical level explanations 69

5.4. TRAMIS user interface and monitoring 70
5.4.1. User level explanations 70
5.4.2. Technical level explanations 71

Chapter 6
A generalization/specialization construct in TRAMIS: the category 72

6.1. The category and TRAMIS architecture 72

6.2. The category in TRAMIS model 73
6.2.1. User level explanations 73
A. The category 73
B. Inheritance 76
C. Categories and naming conventions 77
D. Categories and statistical model 77
E. Category and identifier group 78
F. A new integrity constraint: the global cardinality 78
6.2.2. Technical level explanations 79

6.3. The category and TRAMIS processors 81
6.3.1. User level explanations 81
A. Consultation 81
B. Modifications 81
C. Category and model compliance checking 83
D. Category and import/export 83
E. Category and executable schema generation 84
F. Category and report generation 84

G. Transformations of a category 85
6.3.2. Technical level explanations 92

6.4. The category in TRAMIS user interface 93
6.4.1. User level explanations 93
6.4.2. Technical level explanations 93

Conclusion 94
Main ideas 94
Other works 95
Evaluation 95
Future work 96

References I
List of references I
Classification of references XIV

Acronyms XVII

Appendix A
Reference example .....A

Appendix B
The GER model C

Appendix C
Extract of TRAMIS user interface monitoring .....G

Appendix D
Extract of TRAMIS user interface monitoring including the category .....K

Appendix E
ADL algorithms concerning the category .....Q

Addendum
A more complete description of the ADL algorithms concerning the category .....R

Introduction

Background

Modelling is the major task in computer science. Such a task uses *abstractions* to deal with the complexity of the real world, a model of which computer application must be. We abstract from details, in order to reduce that complexity and to keep only what is relevant to the application.

Database management systems (*DBMSs*) are typically used for the management of data stored in the *database* which stand for 'facts' in the reality, the so-called *Universe of Discourse* (*UoD*). As every software component, the database must be organized in a well-structured fashion. This structure is explained in the *database schema*, in terms of the constructs of a *data(base) model*. Building this description is called *database design*. A data model has to meet two different objectives: to record technical information about the storage of data, but also the semantics of the reality. The first objective has always been obvious, the importance of the second emerged only in the early seventies. Other intermediate aims may be stressed, for example, the user view of data. This multiplicity of objectives leads to the introduction of *hierarchies of descriptions*.

The first level and the most critical one, called *conceptual modelling*, aims at describing the *semantics* of data. Such specifications are closer to human perception than to the computer aspects. Therefore *conceptual models*, i.e. database models used for conceptual modelling, have to include structures matching with abstraction mechanisms used in human mind.

The recent years have seen the emergence of *computer-aided software engineering* (*CASE*) tools which are helping the designer to build software components, according to different methodologies. In particular, the *database design workbenches* offer one or several models and are tackling different aspects of the design of a database.

Objective and motivation

In this dissertation, we focus on the *generalization/specialization abstraction structures*, trying to provide a theoretical study, but also examining its integration in a database design workbench.

The generalization/specialization abstraction structures offer a powerful specification tool, but also an implementation tool. The idea of the generalization mechanism is to define a class of objects describing the objects of other classes in avoiding details. The specialization mechanism implies the definition of classes which are more specific than a given class. We focus on its use in the conceptual phase of the design of a database, although we also outline its interests in the whole development of software and in three major areas of computer science (programming, artificial intelligence and database) which have been confronted in the recent years.

Currently, it is recognized that methodologies 'need' the use of software tools. We integrate the generalization/specialization construct in the model of *TRAMIS*, a database design workbench developed at the Institut d'Informatique and marketed by Concis. This tool, based on the *toolkit* approach, supports a single *entity-relationship* (*E-R*) model for all steps of database design, and provides a user interface using *windows*. We therefore study how a generalization/specialization construct can be incorporated in the model (extending by the same way the E-R model), how existing processors must be modified and what the new functions are, and finally how that construct can be introduced in the user interface.

Related works

Generalization/specialization constructs are often quoted in the literature. However no consensus seems to appear. Some studies [Coll 88] [Hain 89d] [Hull 87b] [Peck 88] [Spac 89] have been devoted to their analysis for semantic modelling, [Tour 86] study it for artificial intelligence, and all works on object-oriented programming deal with it more or less thoroughly.

On the other hand, studies on the problem of integration in a database design is less proposed. Only papers explaining different database design workbench philosophies are published. We didn't discover any

'methodological' book stressing the process of developing or extending a database design workbench.

Main ideas

From the theoretical analysis, we derive the main facets of generalization/specialization structures, i.e. *class inclusion dependencies* and the *inheritance inference rules*.

For the practical aspect, we emphasize that the construct should provide a *useful*, *minimal* and *consistent* framework for generalization/specialization structures which is compliant to the 'philosophy' of TRAMIS. *Transformations* are given a special care: we provide transformation not only for the representation of categories by basic constructs, but also for their manipulations.

Outline

In chapter 1, we present the concepts of abstraction mechanisms and define the most frequently used mechanisms. An overview of the generalization/specialization mechanism during application development lifecycle, and in three major fields of computer science is proposed too. Chapter 2 recalls some fundamental aspects of database design, especially at the conceptual level. In chapter 3, we analyse, within the GER framework, the different facets of generalization/specialization constructs of most data models, trying to understand their usefulness and applicability. Chapter 4 overviews how generalization/specialization constructs fit in the data modelling process: we explain how and when generalization/specialization should be introduced in a schema, and we briefly describe transformations concerning them. Chapter 5 explains TRAMIS, the database design workbench developed at the Institut d'Informatique. In chapter 6, we investigate the problem of introducing a generalization/specialization construct in TRAMIS. This construct, called the category, is explained as a component of the TRAMIS model; we then analyse its impact on the different processors and show how it can be integrated in the user interface. The conclusion is concerned with a summary of the main results, with a discussion of our work, and with possible future researchs on the study of generalization/specialization and on the development of workbenches. Appendix A explains the reference example. In appendix B, we give an overview of the GER model which is used as framework in the theoretical study. Appendix C presents the principle of the monitoring of the TRAMIS user interface,

while appendix D shows its monitoring when the generalization/specialization construct is introduced. In appendix E, we provide a set of ADL algorithms working on the GAM schema of the specification database which concern the proposed generalization/specialization construct.

Note to the reader

The reader is supposed to be acquainted with the E-R approach and with the traditional relational model. We moreover assume that he is familiar with database design problems.

When no consensus appears in the literature about a question, we provide our viewpoint when necessary.

At the beginning of each chapter and paragraph, the reader may find a summary of its contents.

Chapter 1

The generalization/specialization abstraction mechanism

We present here the abstraction mechanisms, and focus on one mechanism, the generalization/specialization, trying to examine its applicability and to study its usefulness in the software development lifecycle, and in three major areas of computer science, namely artificial intelligence, programming, and databases.

1.1. Abstraction mechanisms

Programming is one of the major tasks in computer science; by *programming*, we mean conceiving a model of a certain part of the reality according to a certain formalism. It is therefore interesting to have a closer look at the modelling process, and try to understand the mental mechanisms involved in each modelling task better.

The basic process employed whereby a model of the reality is designed is known *abstraction* [Davi 89]: it involves the selective emphasis of details; the final result of applying abstraction is that the elaborated model stands for, in a meaningful way, those aspects of reality being of interest for the human 'observer'. Abstraction is a mind-structuring process in which one omits details about a fact, in order to reduce the complexity of the real world, and so to be able to build an 'operational' model of it.

There are different kinds of abstraction mechanisms, leading to hierarchies of abstractions, because there are too many details in the reality for a single abstraction to be intellectually manageable [Smit 77]. Essentially there have been much work around four abstraction mechanisms [Boda 89] [Brod 81] [Brod 84a] [Codd 79] [Davi 89] [Hamm 81] [Hull 87b] [Matt 88] [Mylo 84] [Ridj 84] [Smit 77] [Sowa 84], at least in modelling techniques:

- classification;
- aggregation;

- association;
- generalization.

1.1.1. Four main abstraction mechanisms

A. Classification

The *classification* consists in the grouping of objects sharing common characteristics into a *class* (also called *set* or *type*) over which uniform conventions hold [Borg 84]. It is a form of abstraction in which a collection of objects is considered as a higher level object class which is a precise characterization of all properties shared by each object in the collection. We say that an object is an *instance* of a class if it has the properties defined in the class [Ridj 84]. Classification represents then an *is-instance-of* relation between an object and a class. By definition [Boda 89], each instance inherits the properties of the classes. The inverse is *instanciation*.

For example, an object class book that has properties title and ISBN code may have as instance the object with property values *On conceptual modelling* and *0-387-90842-0*.

This is a fundamental mechanism as it allows an economy in the description of the semantics: we can describe the modelled domain in *intension*, and not in *extension*. It is encountered in many computer science applications to identify, classify and describe objects in terms of object classes [Ridj 84].

We may note that a class can be considered as an object itself, and so belong to a class too (called a *meta-class*), and so on. The class 'aspect' is not an intrinsic property of an object, as explained in [Hain 89d].

B. Aggregation

Aggregation consists in the treatment of a collection of objects as a single concept [Borg 84]. Therefore it implies that an object has components as part of its structure that are themselves objects of interest in the UoD. In other words, it is a form of abstraction in which a relationship between component objects is considered as a higher level aggregate object [Ridj 84]. This is the *is-part-of* relation. Aggregation supports

upward inheritance, i.e. the aggregate can receive the properties of its components. Its inverse is *disaggregation*.

For instance, a company car may be the aggregate of component wheels, engine, etc.

Aggregation is useful as an abstraction mechanism in that it gradually makes the composition of an object visible, and depicts how the components which comprise the object relate to each other and to the object as a whole [Davi 89]. It is also widely present in programming languages and database models (where *object identity* [1] has been introduced to complete it).

Again, we may note that an aggregate object may itself be a component object of another aggregate object.

C. Association

Association is a form of abstraction in which a collection of member objects is considered as a higher level set object [Brod 81]. Association (also called *grouping*) is based on the *is-member-of* relation: this form of classification relates the instances of a lower level type to the instances of the higher level type containing classes to which the instances of the lower level type do belong [Davi 89].

For example, the set trade-union is an association of employee members.

It allows to create an object set from members of a class which verify a same criterion [Boda 89]. It is a less recognized mechanism. According to [Ridj 84], it emphasizes set oriented design as a special case of aggregation.

D. Generalization

Generalization consists in the extraction from one or more given classes of a more general class that captures commonalities but suppresses some of the detailed differences in the description of the given class [Borg 84]. We may notice that it introduces an inclusion dependency: one or several sets of objects are considered, at a higher level of abstraction,

[1] The idea of *object identity* is that an object has an identity (that distinguishes this instance from all other instances) and a substance (the properties that hold for the object and can be discovered by investigation of the object) [Zill 84].

as a set which includes them. This is the *is-a* relation. The inverse is called *specialization.* Inheritance is associated with generalization.

For instance, the person class may be a generalization of the author and book reviewer classes.

We shall analyse it thoroughly in this work. It has been largely used but there are many variations on the ideas. Incidentally, Smith and Smith [Smit 77] felt the need to develop a philosophical position on the nature and representation of generalizations.

A class which results from the generalization of other classes may itself be generalized.

1.1.2. Hierarchies of abstractions

Actually, all abstractions are *orthogonal* in the sense that none of them can be derived from others [Nava 88]. An object (or a class) can be a manifestation of more than one abstraction provided that orthogonal nature of abstractions. Classification is a relation between a class and its instances, whereas aggregation, and association are used between classes to 'characterize' their instances, and generalization is typically for classes. The class relations are the important ones for modelling. Any class can simultaneously have aggregation, association, and generalization relations with other classes.

All these mechanisms (classification/instanciation, aggregation/disaggregation, association/membership, generalization/specialization) provide techniques for structure modelling. Modelling typically involves the identification of relations of all objects (classes) relevant for the field to modelize. They take advantage of property inheritance, the main richness of which is abstraction (suppression of details), modularization, and consistency since essential properties of an object (or a class) are defined once and are inherited in all relations in which it takes part. Classification supports downward inheritance; aggregation and association support upward inheritance in which properties of the components or members are inherited by the aggregate or set. Generalization supports downward inheritance in which all properties of a generic object class are inherited by each specific object class (but also upward inheritance).

1.2. The generalization/specialization abstraction mechanism in software development lifecycle

It is a clear evidence that a successful software development methodology has to employ as many as possible of the abstraction structures mentioned above, in order to make the modelling task easier.

1.2.1. Software development lifecycle

It is generally recognized that the development of a software, or more globally an information system, goes through different steps before it is operational [Boda 89]. These steps are usually:

- the *requirement analysis* which is concerned with the elaboration of the definition of systems functionalities, performances constraints, ...;
- the *conceptual* or *functional analysis* which develops a functional solution but independent from any technical media;
- the *design of the software architecture*;
- the *implementation* and the *testing*.

1.2.2. Software development based on generalization/specialization

Requirement analysis is largely recognized as being the most critical step in the software lifecycle, since errors on this level may have disastrous effects on the subsequent development steps [Dard 89]. In that level, both users and analysts cooperate to elaborate the requirements. As generalization/specialization structures are largely used in the real world, they will be useful too.

In the conceptual step, generalization hierarchies help the designer to organize the process of gathering details and integrating them into a consistent information system.

Generalization should be used as a cornerstone in designing data-intensive applications [Borg 84]. Software specification methodologies can combine stepwise refinement by decomposition with concept specialization in order to introduce the multitude of details typically associated with large interactive systems [Borg 84]. We shall later analyse the database design aspect, however it should be noted that generalization/specialization is

useful in the 'application' aspect[2] too. Indeed, in the development of the software, we have to define software relations between software components (modules). Among the different relations which are widely used [vLam 88b], the *inheritance* relation typically hides a generalization/specialization structure.

On the implementation level, object-oriented programming languages are becoming more and more important, and they make a wide use of generalization/specialization (see next paragraph) which enhances both reusability and modularization, while reducing clerical errors.

1.3. Generalization/specialization in programming, artificial intelligence, and database areas

At the very beginning of researches on data modelling, it has becomed obvious that this area has a lot of common interests with knowledge representation in artificial intelligence: models of the reality are important for both database and artificial intelligence systems. For some time, it has been a growthing work on the confrontation of these two approaches [Mylo 89] [Sern 85], along with programming languages [Brod 80] [Brod 84a]. For instance, many of the concepts in semantic data models are derived from research in two major areas in computer science [Afsa 86]:

- data abstraction ideas in programming languages (abstract data types);
- knowledge representation concepts in artificial intelligence.

This work about generalization/specialization in database conceptual modelling can certainly take profit from a survey about the exchange of ideas among researchers who are concerned with:

- abstract data types and program specifications techniques, in the programming language domain;
- knowledge representation, in the artificial intelligence community;
- conceptual (semantic) database modelling techniques in the database sytems research area.

[2] We follow the traditional database approach, where there is a distinction between *data* and *applications*. Another trend, namely the *object-oriented approach*, considers objects as the basic building blocks; in this approach generalization/specialization plays an important role too.

1.3.1. Generalization/specialization and object-oriented programming

A. Object orientation

According to the object-oriented paradigm [Amer 87] [Duco 87] [Duco 89] [Halb 87] [Higg 89] [Meye 89], the world contains distinct *objects*. Each object is an instance of a *class* or *type*. Thus, the world is described in terms of abstract data types which are related to each other within *inheritance hierarchies*. These objects have a *state* that can change over time (for several of them). The state of an object is indicated by values of its *attributes*. Each class contains the description of the attributes which are given values for instance objects of that class. The desired behaviour of objects forms a part of the object definition: object behaviour is defined in its *methods*. Objects have an identity within the system and are manipulated through *messages* to which they respond by mapping the message and its method which is then invoked. An object can accept or reject a message sent to it.

Inheritance is a way of increasing re-use and enhancing modularization: if two or more objects of different types share some common properties or behaviour, a new type can be defined that contains the shared properties or behaviour, and the original types are defined as *subtypes* of the newly created type (called the *supertype*). On the other hand, if one recognizes that some instances of a given type could share some specific properties or behaviour, one or more new subtype(s) can be defined. Each subtype inherits the properties and methods of its supertype(s). In addition it may add properties and methods that are unknown to the supertype(s). When a type inherits a method it can choose:

- to inherit the described behaviour;
- to refine that behaviour by replacing the inherited method by its own;
- to disallow the behaviour.

If a type is permitted to have only a single supertype, the system is a *single inheritance* system, otherwise it is a *multiple inheritance* system. Note that multiple inheritance introduces several problems (inheritance and naming conflicts, ...) [Duco 87] [Duco 89].

B. Generalization/specialization and objects

Amid the different features of object-oriented programming languages, there is *inheritance* (generally considered as a necessary, although not sufficient, condition for a language to be considered *object-oriented* [Higg 89]). Inheritance allows classes to share definitions with other classes, and to make that commonality explicit. It is therefore a technique that allows new classes to be built on the top of other less specialized classes rather than be written from scratch. [Halb 87] discusses different ways to use types and inheritance to structure programs.

Note that some types will have a set of instances, whereas others will not. Types that are only intended to provide a common set of properties and operations for their subtypes, but which are not intended to have instances of their own, are called *abstract supertypes* [Halb 87].

1.3.2. Generalization/specialization and artificial intelligence

A. Knowledge representation and manipulation

Artificial intelligence is the study of knowledge representations and their use in language, reasoning, learning, and problem-solving [Sowa 84]. In artificial intelligence systems, representation schemes are used for representing semantics [Alte 90] [Brac 85] [Mylo 84]. One may classify the representation schemes as follows [vLam 88c]:

- standard and non standard logics;
- production rules;
- structured objects (these representation techniques are inspired from associative memory of human beings), i.e. *semantic networks* [Brac 83] [Wood 75] and *frames* [Brac 89] [Fike 85].

Knowledge is processed by means of strategies of searching and reasoning [Alte 90]. Indeed, with conventional database systems, the user must know what to ask for and what to do with results (they are passive, although deductive databases are becoming more active); a knowledge-based system keeps track of the meaning of the data and performs inferences to determine what information is needed even when it has not been explicitely requested (see [Brod 86] for a confrontation of these systems).

In order to obtain new information from existing one, the following types of reasonings may be applied [vLam 88c]:

- formal reasoning (or deductive);
- inductive (or by generalization) reasoning;
- reasoning by analogy;
- procedural reasoning;
- reasoning by default.

B. Generalization/specialization in knowledge representation and reasoning

The structured objects representation schemes make a wide use of inheritance because they reduce repetitions in descriptions and so reduce the number of clerical errors: regularity is exploited by creating abstractions classes. In semantic networks, a privileged relation, the *is-a* relation [Brac 83], incorporates a strict inheritance mechanism (classes are *templates*). Frames, on the other hand, consider classes as *prototypes*, on which default inheritance can be applied [Borg 84]. With multiple inheritance, especially if default inheritance and exceptions are allowed, reasoning (which is part of artificial intelligence systems) has become more difficult to deal with: numerous algorithms have been designed using the shortest path reasoning, searching all inheritance paths or other 'techniques' (some of them have unpredictable behaviour) [Tour 86].

1.3.3. Generalization/specialization and databases

A. Database models

A *database* is a model of an evolving physical world; the state of this model, at a given instant, represents the knowledge that has been acquired from the world [Abri 74]. The data are organized into a structure which is defined by the *database schema*.

A *data(base) model* is a collection of mathematically well-defined constructs and integrity rules helping at the expression of static and dynamic properties of data intensive applications [Brod 84b]. A data model must have a commonly accepted (and useful) interpretation [Date 86]. Indeed, a data model is a formal system (i.e. a system in which a set of precisely defined objects can be manipulated in accordance with, and only in accordance with, a set of precisely defined rules, without any regard for the real world interpretation of those objects and rules).

Since their origin, data models are at the confluence of two distinct necessities: the representation of the reality and the representation of the internal structure of data (for software and hardware). The first group contains *representation models*, while in the second group are the *recording models* [More 80].

If we overview the history of data models, we may distinguish different distinct groups characterized by levels of abstractions within the modelling formalism [Pott 89]:

- the *ancestors*, i.e. file systems;
- the *traditional data models*, i.e. the hierarchical group, the network group [Hain 85] and the relational group [Codd 70] [Delo 82] [Hain 88];
- the *semantic data models*.

Most traditional data models provide essentially means of representing data (they are *computer-oriented* or *record-oriented*). Semantic data models, through the use of abstractions, allow the user to model and view the data on many levels [Peck 88] (they are *user-oriented* or *semantic-oriented*). Thus the objective of designing a semantic data model is to design a higher-level database model that will enable the database designer to naturally and directly incorporate more of the semantics of a database into its schema [Hamm 81].

The paper of Peckham and Maryanski [Peck 88] surveys and compares a representative sample of semantic data models ([Hull 87b] and [Hain 89c] propose the use of a pedagogical model for comparisons). These are analysed for the presence of constructs representing the fulfillment of general semantic modelling goals.

There are (at least) two philosophies in data modelling as to what the real world consists of: just objects, or objects and relationships amid them. In the first case, associations between objects have to be expressed by general references, in the second one, a top-level concept is available [Ditt 90].

Chen's E-R model [Chen 76] is an early semantic data model that unifies features of the traditional models and facilitates the incorporation of semantic information [Peck 88] by attempting to provide multiple abstractions levels. Other semantic models are the binary models [Abri 74]

[Fou 89] [Hain 74] [Hain 86a] and the extensions to the relational model [Codd 79]. Other models follow the functional approach [Ship 81] or the logical approach [Gall 89].

Currently much work is done on the design of new semantic models or on the extensions of existing models.

Facilities for complex object modelling [Hull 87a] are nowadays needed in several database applications like computer-aided design/computer-aided manufacturing (CAD/CAM). Complex objects modelling concerns structural object orientation, i.e. the ability to support structural description of complex objects together with generic manipulation operators, defined for all objects types [Pare 89]: the basic facility needed for such modelling, beyond the concept of object itself, is the ability to specify that an object is represented by a collection of information, decomposable into components; each of these components, in its turn, may be represented by a collection of information, and so on.

The central result of semantic modelling research has been the development of powerful mechanisms for representing the structural aspects of business data. In recent years, however, the attention of database designers has turned towards incorporating the behavioural or dynamic aspects into modelling formalisms (influence of object-oriented programming) [Hull 87].

Another trend is towards hyper-semantic data models and deductive databases. These models acquire knowledge in the form of artificial intelligence concepts, together with concepts of semantic data models [Pott 89]. The hyper-semantic models are then user-oriented and they capture inferential relationships among real world concepts. They allow one to define deductive databases that have the ability to deduce new facts from an arbitrary set of facts and rules, to supersede fact-based database schemes.

B. Generalization/specialization constructs in database models

Generalization/specialization has been introduced in semantic models, but also in implementation models (e.g. in object-oriented DBMSs).

a) Generalization/specialization in semantic models

The concepts of generalization/specialization, leading to subtype/supertype hierarchies, are of particular importance, as far as conceptual

modelling is concerned [Hain 89c]. Developed mainly in semantic representation techniques, they have been adopted in the framework of semantic data models.

As we shall analyze later, the main interests of these concepts are the inheritance inference rules.

We may effectively distinguish three possible interests of generalization/specialization constructs, as far as conceptual modelling is concerned:

- to add more semantics about the UoD in schemata;
- to allow more schema flexibility;
- to allow more conciseness in schemata.

The first and primordial interest is to allow the representation of more semantics about the UoD in the conceptual schema. By definition of the generalization/specialization mechanism, if we have a construct which allows its representation in a schema, we are then able to represent the same object of the UoD under different and complementary aspects, and to deal with these multiple representations. An important remark is that if we allow multiple representations of a same object in the schema, we must be able to manage possible redundancies of representation, and to avoid any contradiction.

The interest, is that it allows to have only one schema which can be read by different persons, according to their viewpoints. Semantic relativism, i.e. the ability to view a same information in different ways depending on the context [Brod 81], is enhanced.

This construct is also used in order to obtain more concise schemata. Generalization may be introduced in order to put together characteristics belonging to several types in order to form another type. We already saw the interest from a *set of instances* viewpoint, while we consider here a *type description* point of view.

b) Generalization/specialization in implementation models

Generalization/specialization constructs are also used in the database models of object-oriented DBMSs. Currently, there are concerns about the real nature of such DBMSs and around their issues. As said by Higgs in his survey [Higg 89], 'it is clear that the research community has

not yet distilled out the essentials of an object-oriented database system to everyone's satisfaction'.

The idea is to apply to database many of the ideas that had success in programming languages (cf. 1.3.1), in order to achieve similar advances, especially for applications coming from engineering, manufacturing, CASE, CAD/CAM, knowledge-based systems, office automation ..., where current database technology seems to fall short: the data model of traditional DBMSs are not expressive enough to effectively model these applications [Higg 89].

Dittrich brings clarification to the object-orientation in the context of DBMSs, and presents issues in his paper [Ditt 90]. He presents a list of features that an ideal object-oriented DBMS should have (according to a number of researchers from six different schools): an object-oriented DBMS is a DBMS with an object-oriented data model. An object-oriented data model is a data model which has, among others, the following characteristic: types (classes) can be organized into hierarchies and thus allow to express that one type is considered as a subtype of another one, i.e. that it is specified in more details (with respect to structures and operations) than the supertype. Along with type hierarchies goes the concept of inheritance in that a subtype inherit the properties (structures and operations) from the supertype in addition to its proper ones. Of course, inheritance propagates all the way to the top of the hierarchy if there are more than just two levels. In summary, with types hierarchies and the inheritance mechanism, more semantics can be expressed than without, it introduces an additional modelling discipline (refinement), and it may save coding efforts because operations of specific classes need not always to be recoded. Closely connected are the concepts of overriding, overloading and late binding.

We may also note that [SuYS 86] proposes the use of generalization/specialization for modelling partitioned and replicated databases in distributed databases.

Chapter 2

Database design

In this chapter, we recall a few concepts concerning database design. These explanations aim at providing the reader with our point of view (especially when no consensus appears or when different approaches are possible).

2.1. Database design methodology

We shall first locate the database as a component of the information system, and present the idea of design methodologies.

2.1.1. The information system and the database

The general model [Boda 89] of an *information system* is represented in figure 2.1. We consider that an application receives inputs, provided by its environment (i.e. a user or another application), and computes, according to its specification, some outputs that are given to the environment. This computation consumes ressources and there is interaction with the database.

Let us precise the role of the *database*. Computer applications aim at providing an enterprise with some automated work which can be of crucial importance for that enterprise. Many of these applications interact with the database, the memory of the information system. It consists in a collection of data standing for information in the real world. We recognize two roles of a database [Hain 86a]:

- it is a *correct*, a *reliable model* of a part of the real world;
- it must be an *efficient data server*.

2.1.2. Database design methodology

We have already mentioned that as any software 'component', the structure of the database must be known to the computer. The global user's view of a database, i.e. its *schema*, is specified in terms of a database

description and structuring formalism, called a *data(base) model*. The process during which the database is built is called *database design*.

It is now recognized that information systems (and consequently databases) are not built in an arbitrary way but rather according to a *methodology* (a survey of methodologies can be found in [Roll 89]). Bodart and Pigneur [Boda 89] explain that any methodology is based on a set of *models*, proposes different *steps and rules* which are working with the help of *software tools*.

A. Design steps

One inherent problem in modelling any subset of the real world is the difference between man's perception of the enterprise and the need of the computer to organize data in a particular way, with efficient storage and performance constraints.

This gave rise to *modelling levels*. [Hain 86a] provides an overview of the evolution of these aspects. In this work, we consider that the design of a database is organized as follows:

- the *conceptual modelling* step produces a conceptual schema specifying semantic structures that the database will contain in order to represent the UoD;
- the *logical modelling* step aims at producing a schema containing all the semantics expressed in the conceptual schema, but describing the logical accesses needs of application and being consistent with a DBMS;
- the *physical design* step produces a description of the database which is both operational and efficient.

In this dissertation, we essentially focus on conceptual aspects (and the transition towards logical design). They really important in the database design process [Plet 89].

B. Data models

In most methodologies, to each step corresponds a different model (see chapter 1 for the survey of data models). For instance, conceptual specifications are written with the E-R model, logical specifications use a binary model, and the DBMS, physical schema is a relational schema. The practical use of semantic data models has often been limited to the design

of record-based schemata, i.e. the DBMS data models are typically record-oriented. Therefore, high-level structures of user-oriented models expressed with a semantic model are mapped to record-oriented structures. Some methodologies go directly from the conceptual model to the DBMS model which is used for both the logical and physical steps [Berm 86] [Saka 83] [Teor 86] while others use an intermediate model for the logical step [Hain 86a].

Such design methodologies are essentially based on a well-defined set of *transformations* used for these model mappings.

Hainaut devoted much work to transformations: [Hain 81] [Hain 86b] [Hain 87] [Hain 89a] [Hain 89b] [Hain 89c] [Hain 89d]. We take the following definition of *semantic preserving* (or *reversible*) *schema transformations* from [Hain 89b]: transforming a source schema produces a new schema which has some sort of equivalence with the first one (we are interested in a semantic equivalence), but satisfies some sorts of constraints the first one does not meet. A reversible transformation transforms data without loss nor noise in the database, i.e. so that an inverse transformation allows to retrieve the initial state. Transformations concerning generalization/specialization structures will be analysed in subsequent chapters.

C. Computer-aided design

CASE tools are becoming important for the efficient management of all steps of information systems lifecycles: they reduce the 'clerical' work and facilitate coordination among persons concerned by the design (an analysis may be found in [vLam 82]).

Currently, there are more and more database design workbenches. They are either a component of a CASE tool [Cata 88], or subclasses of them [Hain 89d]. While every software tool rests on a lifecycle model, some of them compel the user with a rigid sequence of operations and others offer a toolkit approach. Some of them are front-end tools only, while others go through all steps of their lifecycle model. Some of them are 'traditional' programs [Conc 90b] [Lepr 86] [Tuch 90], and others offer an expert system approach [Bouz 84] [Bouz 86] [Bria 85] [Cive 88] [Mann 88] [Roll 86] [Spri 88] [Stor 88].

In chapters 5 and 6, we shall be interested in an independent database design workbench, based on the toolkit approach, going through all steps of database design, and being a 'traditional' program.

2.2. Conceptual modelling

After having browsed the idea of database design, we explain here the principle of the conceptual modelling phase. We first recall the 'link' between the UoD and the database to understand the conceptualization process better. Then we say a few words on conceptual models, and we overview the activities involved in the building of a conceptual schema.

2.2.1. The UoD and the database

A. The UoD

The UoD, as already mentioned, is a part of the real world which is pertinent to the application (i.e. we want to model). In our example, it refers to a library[1]. We will assume, according to the *object-association* philosophy, that the UoD contains individual components (the objects) which are in relation with each others, and are characterized by properties.

If we were asked to explain the UoD or if we think about it, we should certainly not quote all the objects and relations present at a given time: we should, consciously or not, use abstraction mechanisms. In other words, we structure our mind by introducing abstraction levels in order to reduce the complexity of the real world. These abstraction mechanisms have already been explained in chapter 1. However we have to precise that during the conceptual modelling process for the description of structural properties[2] of the real world, the classification mechanism has a 'special' role.

Indeed, the UoD is composed of two parts [Fouc 89]:

- the *individual system*, where objects are living;
- the *abstraction system*, which contains the rules describing the behaviour of objects (it has no physical existence in the real world, it results only from a mental, abstraction process).

B. The database

The database (a part of the information system) can be defined as a formal representation of the UoD.

[1] The following examples are taken from a common case study explained in appendix A.

[2] This work focuses only on static aspects of information modelling, leaving aside query and manipulation languages, and behavioural aspects.

Of course, as we distinguished two parts in the UoD, we shall find them in the database[3]:

- the *conceptual schema* is the result of the formalization of the abstraction system;
- the *occurrence schema*[4] contains the representation of the real facts (i.e. what is in the object system of the UoD).

But we have to remark that the role of the abstraction system of the UoD is to describe the objects and relations, while the conceptual schema predetermines all possible states and transitions allowed in the occurrence schema, according to rules which explain what is happening in the UoD.

Keep in mind too that we assume that a single fact in the UoD is represented at one and only one place in the database (i.e. that each object of the UoD maps onto one and only one element in the occurrence schema [Mcle 80]).

The conceptual schema describes the semantic structures of data (or according to another interpretation, the structures of the UoD described by data) [Hain 89d]. We assume that such a description is independent from any 'technical media' and easy to understand by 'everybody' (users, designers, analysts, programmers). Indeed its objective is threefold:

- to help the user, the designer and the analyst to express meaning of data, i.e. information, which is relevant to the application (*communication* viewpoint);
- to help the builders of the system organize data correctly according to their semantics (*computer application* viewpoint);
- to help give the users the precise definition of the information they manipulate, and their conditions of use (*exploitation* viewpoint).

2.2.2. The conceptualization or conceptual modelling process

The conceptual designer first acquires knowledge of facts (and feasible facts) in the UoD by observation sessions, interviews, analysis of documents, reverse engineering, ... Then, by abstraction, he will speak in terms of classes and consistency rules. Finally there is the formalization

[3] The dichotomy between classes and instances is an axiom in databases.
[4] Often, the occurrence schema is called the *database*.

of that abstract knowledge (using a conceptual model) in order to build the conceptual schema. In summary, we may distinguish three steps: *observation*, *abstraction* and *formalization*.

In the following, we shall be interested in the formalization step, i.e. the building of a conceptual schema, especially, with the idea of building a conceptual model (more precisely enhancing the capabilities of an existing one) in order to simplify the expression of abstractions in the schema and so facilitate that formalization step.

2.2.3. Conceptual models

Their aim is to be easier for the designer to draw 'natural' structures and for the user to manipulate concepts that are already familiar for him [Borg 85]. Our hypothesis is that the better a model is able to reflect the abstraction processes, the closer the user's way of thinking it will follow and the easier it will be to use. Therefore semantic data models (see chapter 1) should be used.

In the conceptual modelling phase it is especially important to stress objects pertaining to the application domain, and to try to eliminate as much as possible of the noise introduced by the constructs of the specific model used or by its incapacity to express subtilities, necessary for the enterprise [More 88].

2.2.4. Building a conceptual schema

They are different alternative *schema design* approaches [Nava 88b]:

- design of the *entire schema* as a single activity;
- design of a first schema, and obtention of the final result schema by a process of *schema refinement and restructuring*;
- design of *component schemata* and then *integration* of these schemata.

Other activities concern *normalization* [Boda 89] [Lenz 89] [Tard 88], i.e. the checking of the completeness, consistency, stability, conciseness of the conceptual schema, and *view extraction* (for documentation purposes) [Bati 88] [Teor 89] [Urba 87] [Verm 83]. When extended constructs are proposed, mappings towards basic concepts are often done (another approach consists in the direct mapping of extended concepts in implementation

models constructs). *View integration* is studied, among others, in [Bati 86], [Cive 88], [Mann 88], and [Nava 88b].

All these activities use three different 'operators': schema modification, verification of a schema according to rules, and transformations of schemata. They will be studied in this dissertation, as far as generalization/specialization structures are concerned.

Chapter 3

Generalization/specialization structures in database models

In chapter 1, we saw that generalization/specialization is a powerful abstraction mechanism, and among others, useful in a database conceptual model. The aim of this chapter is to study the different constructs and their related elements proposed in the literature about this mechanism. This analysis will help us to define a generalization/specialization construct for TRAMIS.

Indeed, much attention has been paid for generalization/specialization concepts during the last decade in data models. It is generally recognized that Smith and Smith [Smit 77] introduced it in a data model. Incidentally, in his paper introducing the E-R model [Chen 76], Chen hinted at subsetting of entity types was possible. Codd proposed a generalization/specialization construct in its extensions of the relational model [Codd 79]. The generalization/specialization concepts are now available in almost all data models in most cases as one or several constructs, but sometimes as integrity constraints (see [Boda 89], for example) and with more or less flexibility. There are actually many variations around both their interpretation, the way they are used and the vocabulary, such that the impression of unanimous consensus disappears as soon as one takes a closer look on the different proposals; it then becomes difficult to get a clear idea of what is the basic significance of the concept, and of what is a particular choice among different possibilities [Spac 89].

A few studies have been devoted to the generalization/specialization concept:

- [Coll 88], [Hain 89d] and [Spac 89] study it as an extension of an E-R model;
- in [Hull 87b] and [Peck 88], it is considered as one of the characteristics of semantic data models.

The examples in the following are referring to the library UoD explained in appendix A. The GER model (outlined in appendix B) is used as

a framework for the explanation of the generalization/specialization concepts.

3.1. The generalization/specialization basic block: the *is-a* relation

Before getting in more details about the *is-a* relation, let us recall the basic notion of generalization/specialization in data modelling as explained in chapter 1. The constructs of generalization/specialization allow everyone to specify that a class of objects contains the objects of another class (inclusion dependency), and the first class describes them in a more global, in a less precise way than the latter class does (details avoidance). Such a mechanism allows the recording of abstraction levels in the conceptual schema; they are often used in the conceptualization process, and allow one to master the numerous characteristics of an object in the UoD (and so capture more semantics). Moreover, this allows some kind of view 'reconciliation': an object may be seen under one or another (non contradictory) aspect, according to a viewpoint. Schema conciseness is enhanced too when this mechanism is used.

3.1.1. The *is-a* relation

A. Definitions

Concepts for generalization/specialization in data models are most frequently introduced, implicitly or not, through the *is-a* relation between entity types. But the different works are based on rather different, intuitive or more formal, meanings of this relation [Hull 87b]. Therefore, we would like to clarify a little by overviewing different definitions.

One of the most used definitions states that entity type ES *is-a* entity type EG if entity domain of ES is a subdomain of EG entity domain: the *is-a* relation is indeed an inclusion dependency [Hain 89d] [Spac 89] [Hull 87b] [Czed 90]. But these authors usually do not allow to say that ES *is-a* ES; however an inclusion dependency is reflexive. Therefore we think that behind the inclusion dependency there is another characteristic of the *is-a* relation: something we might call a *more concrete* relation.

And here, we follow Collart and Joris' viewpoint [Coll 88]: the *is-a* relation is indeed the conjunction of an inclusion relation and the *more concrete* relation. Note however that it is really difficult to formally

define the *more concrete* relation, as it 'goes out' of the formal system. We might say that ES is more concrete than EG if it describes objects in a more concrete way than EG does. [Davi 89] says that an *is-a* relation can be defined between two entity types where one type is the more general one, and the other is the more specific one. According to this approach, ES *is-a* EG if:

- ES entity domain is a subdomain of EG entity domain;
- and the structural and behavioural properties described by ES are more concrete, more specialized than those described by EG.

Another approach takes a closer look at the *more concrete* relation; they 'materialize' it: ES *is-a* EG if ES entity domain is a subdomain of EG entity domain, and all attributes and relationship of EG are also common to ES [Berg 88] [Ceri 81]. They include in a definition what others use as an inference rule (see the *inheritance mechanism* below).

In a similar direction, some authors propose an *is-a* relation as a type constructor: one creates subtypes to make them inherit (and depend upon) the information from the supertype [Simo 89]. The focus is on the type descriptive aspect. The key idea behind the use of the *is-a* relationship is that types share information; avoiding duplication of that information seems to be an important factor in reducing the human cognitive load as well as the use of processing ressources [Simo 89]. Le and Peugeot [LePe 88] introduced an *is-a* relation defining only property inheritance (the *weak inheritance* relation).

It might also be interesting to quote the concepts of *base entity type* and *non base entity type*, with concern to generalization/specialization. A base entity type is defined independently of all other types in the conceptual schema; in our GER framework this means that their entity domain is a basic one. They are mutually disjoint in that every entity is an instance of exactly one base entity type. Of course, at some level of abstraction, all entities are members of entity domain ENTITIES. Non-base entity types are defined in terms of other entity types by inter-types relations: their domain can be declared a subset of a constructed domain [Hamm 81]. The *is-a* relation is such an inter-type connection: specific entity types arise then as derived types. Note that this approach puts in evidence a top-down design of types [Qian 85].

B. Example

In our reference example, the sentence 'Books, journal papers and conference papers are particular kinds of publications' typically hides *is-a* relations. We can represent its semantics by entity types BOOK, CONFERENCE_PAPER, JOURNAL_PAPER, and PUBLICATION; and by specifying that BOOK *is-a* PUBLICATION, CONFERENCE_PAPER *is-a* PUBLICATION, JOURNAL_PAPER *is-a* PUBLICATION, as depicted in schema 3.1[1].

C. Discussion

In the following, we will use Collart and Joris' definition. Indeed, these two characteristics of the *is-a* relation let see the two main advantages of generalization/specialization (from the semantic viewpoint) [Coll 88]:

- an object of the real world may be described in different manners under different viewpoints (as a consequence, this introduces more dynamics in the model: an entity belongs to one or another entity type, for instance by changing of specific entity type);
- different abstraction levels are introduced into a conceptual schema.

We must also be aware that the *is-a* relation is a higher order relation, not between individuals, but between types of individuals [Sowa 84].

Amid the *is-a* relations, some of them are 'natural', i.e. they relate to the 'essence' of the entities, while others are 'role depending' because they depend on accidental relationship to some other entities [Sowa 84]. We can also consider *is-a* relations for 'abbreviation purposes' when they are introduced in order to avoid the definition of a property valuable for different entity types.

3.1.2. The *is-a* graph

A. Properties of the *is-a* relation

If we are considering the *is-a* relation, we may define the following properties.

[1] Note the GER expression of is-a relations.

a) Irreflexivity

The *is-a* relation between ES and EG is concerned by an inclusion dependency: the domain of ES is included in the domain of EG. Sets theory says that this relation is reflexive ($A \subseteq A$). But the definition of the *is-a* relation has a second 'item': the *more concrete* relation which is irreflexive. Hence, the *is-a* relation is *irreflexive.*

b) Antisymmetricity

For the same reason, we cannot have E_1 *is-a* E_2 and E_2 *is-a* E_1. Thus, the *is-a* relation is *antisymmetrical.*

c) Transitivity

Both set inclusion and the *more concrete* relations are transitive. Suppose that E_1 *is-a* E_2 *is-a* E_3, then we also have E_1 *is-a* E_3; the latter relation is non-primitive: it is deducible from the two other specified relations. This is the first inference rule related to *is-a* relations.

B. Definitions

Before following, we shall precise our definitions:

- if ES *is-a* EG then ES is the *specific entity type* and EG is the *generic entity type*[2];
- if this relation is a primitive one, then ES is a *direct specific entity type* of ES, and EG a *direct generic entity type* of ES, otherwise they are respectively *indirect specific entity type* and *indirect generic entity type.*

If ES *is-a* EG_1 and ES *is-a* EG_2, then ES is called a *common specific entity type* of EG_1 and EG_2. If ES_1 *is-a* EG and ES_2 *is-a* EG, then EG is called a *common generic entity type* of ES_1 and ES_2.

C. Properties of the *is-a* graph

To conclude, we can say that the *is-a* relation forms a *strict partial order.* Therefore, the graph formed by all entity types of a schema and *is-a* relations cannot include any cycle: it is an *acyclic-directed graph.*

[2] *In the literature, a specific entity type is also called subtype, specialization, child entity type, subset, or subclass. Supertype, generalization, parent entity type, superset, or superclass are synonyms for generic entity type.*

However it is not connected since there is not necessarily a relation between each couple of entity types. The term *hierarchy* is often used indiscriminately for any partial order [Sowa 84][3]. We have seen that the *is-a* relation is transitive. It is customary to specify the primitive *is-a* relations explicitely and view the relations due to transitivity as specified implicitely (as they do not bring more information).

D. Discussion

When introducing a new construct in a model, one has to define it (local aspect) and to precise the global combination rules. Several authors do not explicitely state global rules, but imply them in the definition of the underlying constructs (see [Hamm 81], for example). Let us now overview some generally specified rules.

A first rule involves directed graphs. It is generally recognized that a generic entity type may have several specific entity types, i.e. *multiple specialization* is allowed. On the other hand, *single generalization* (i.e. an entity type is the specialization of at most one generic entity type) is often required for the simplicity of its management [Boda 89]: in such a case, the *is-a* graphs are trees. If multiple generalization is allowed, we have to deal with 'general' acyclic-directed graphs. For example, one can use multiple generalization to specify that a literary figure is both an author and a reviewer (see schema 3.2). To restrict to single generalization is more restrictive in the sense that we are missing a power of expression; however we shall see that it is sometimes possible to specify with single generalization what is specified with multiple generalization. The drawback of multiple generalizations is that they may introduce lots of ambiguities or contradictions in a graph. For instance, we have seen that the *is-a* relation is transitive, and that transitive links are not introduced into the schema. But what is really a transitive link? E_1 *is-a* E_2, E_2 *is-a* E_3, E_3 *is-a* E_4, and E_1 *is-a* E_5, E_5 *is-a* E_4 are not redundant, especially if we agree that these two assertions are relevant to two different viewpoints about the UoD. On the other hand, E1 *is-a* E3 is redundant with E1 *is-a* E2, E_2 *is-a* E3.

In certain models, it is required that the *is-a* hierarchy forms a *lattice*, i.e. each schema has an entity type which is a common specific entity type of all entity types (the *absurd entity type*, as no actual

[3] We shall also use it with that interpretation.

entity could ever be an instance of that type, since it would be 'everything', all at the same time), and a common generic entity type of all entity types (the *universal entity type* in the GER model).

3.2. Inheritance

As noted earlier, it may be the case that an entity belongs to more than one entity type, i.e. when *is-a* relations are specified: entity types do share entities. As a consequence, they can share descriptive information.

3.2.1. The inheritance inference rule

Since every occurrence of a specific entity type is also occurrence of the generic entity type, that occurrence possesses values for the characteristics (attributes, roles and relationship types) defined in the generic entity type; conversely, some occurrences of the generic entity type have values for characteristics of the specific entity type. This means that an entity type belonging to an *is-a* hierarchy possesses only some of its characteristics; the others may be inferred from its participation to the hierarchy: they are inherited from another entity type. The value for an entity e of ES for an inherited characteristic is simply the value of e when it is viewed as a member of EG [Hamm 81]. For example, schema 3.3 represents the fact that a publication is identified by its ISBN code, is characterized by a title, a topic, and is written by authors; it is also specified that a BOOK is a specific entity type of PUBLICATION. Therefore, one can speak, thanks to inheritance, of the TITLE of a BOOK, and that BOOKs are written by AUTHORs.

Such a mechanism is called *inheritance*. Inheritance is, by definition, the fact that one completes the description of a real object seen through the generic or specific aspect by characteristics respectively specific or generic. Inheritance concerns the descriptive aspect (schema design) but also access to information: at query time, one may access to the information of an entity seen under the specific or generic aspect. In conclusion, inheritance is a reading technique (at schema design time) and a query technique (at manipulation time). We shall analyse now the reading technique, which especially interests us.

With the explanation above, one understands easily that inheritance allows the building of more concise schemata: it is not necessary to repeat

attributes, for instance, of the generic entity type in the specific entity type. Note that we may use this 'tool' as a modelling abbreviation technique, more or less independently of any generalization/specialization consideration. It also allows the building of schemata which are more flexible (see below). As a consequence, the relevant characteristics of UoD objects are not only described in one entity type but may be scattered amid entity types belonging to a same *is-a* hierarchy.

Inheritance is a technique based on an inference rule: this is the second rule, as far as generalization/specification constructs are concerned.

3.2.2. Downward and upward inheritance

A. Definitions

In the literature, there are two tendencies about the inheritance reading technique:

- most of the authors present inheritance from the generic entity type to the specific entity type, i.e. *downward inheritance* (in most semantic data models, each attribute defined on a generic entity type is automatically defined on the specific entity type; it is also generally true that a specific entity type may have attributes not shared by the generic entity types [Hull 87b] [Saka 83]);
- a minority of authors also present *upward inheritance* from the specific entity type to the generic entity type [Carb 80].

In most semantic models, inheritance inference is supported in an anti-symmetrical form: characteristics of the generic entity type also applies to instances of the specific entity type [LePe 88], but none of the characteristics defined for the specific entity type are defined for an instance seen as a generic entity. Other models (the FACT model, for instance [Sacc 88]) support a less restrictive inference rule: all characteristics defined on any entity type to which an instance belongs are defined for that instance no matter its current 'role' is. The latter rule means that when the user or the reader of the schema 'selects' a level of abstraction, he may lose detail on the role but he does not lose information.

We propose to define two primitive inheritance operators: *upward inheritance* and *downward inheritance*. To apply downward inheritance between a generic entity type and a specific entity type consists (from the descriptive viewpoint) in completing characteristics of the specific entity type by characteristics of the generic entity type. To apply upward inheritance between a specific entity type and a generic entity type consists in putting characteristics of the specific entity type in the description of the generic entity type (but optionality has to be specified since only entities which also are specific possess values for these characteristics).

These operators can be combined to offer more complex inheritance inferences; for instance *sideway* inheritance [Carb 80] [Coll 88] consists in applying upward inheritance and then downward inheritance.

Let us now precise what may be inherited. We have seen that inheritance consists in completing the definition of another entity type (belonging to a same *is-a* hierarchy). Therefore, attributes and roles (and the relationship in which they appear) may be inherited. Downward inherited attributes keep their name (if no naming conflict arises), their properties and domain of values, while roles keep their name and cardinalities, and the relationship keep their name, their degree, their roles, and their attributes. Note that key constraints are kept too. Elements inherited in an upward fashion keep their characteristics too, except that the minimum cardinality of an attribute or a role is set to zero, and that key constraints are not standing [Coll 88].

B. Example

Let us consider schema 3.4 representing the fragment of our UoD concerning persons, authors, reviewers and their characteristics. Downward inheritance allows us to speak of the NAME of an AUTHOR. With upward inheritance, we can consider the STIPEND of a PERSON (that attribute may then take the *null* value). If we speak of the PUBLICATION written by a REVIEWER, we apply sideway inheritance: only REVIEWERs who are also AUTHORs have written a book (such instances may exist since no disjunction constraint - see next paragraph - has been specified).

C. Discussion

Authors who reject upward inheritance argue that, if we are interested in an object in its generic form, this means that we 'are' on an

abstraction level so that details from the specific entity type are not relevant. Authors allowing both upward and downward inheritance argue that no matter of the generic or specific approach of the object is, its description should be as complete as possible. In this sense, upward inheritance might be carefully applied [Coll 88]. Upward inheritance allows more schema flexibility (the 'view definition' mechanism is enhanced) and is useful in mapping and view integration activities of database design.

Usually, inheritance is presented as a side-effect, and is sometimes included in the definition of the *is-a* relation itself. Another approach consists in giving the designer (and the user) entire control about the information he wants to specify and to see: inheritance applies only if explicitely stated. This means that we separate descriptive phases from manipulation phases, in such a way that the definition of an *is-a* relation between two entity types is somewhat independent from the fact that someone wants to see more or less information about these entity types. This approach seems to be simpler for users to understand and for designers to implement [Spac 89].

It is straightforward that inheritance is transitive; therefore an entity may inherit an inherited attribute of a generic entity type.

The advantages of characteristics inheritance (especially downward inheritance) are abstraction, modularity and consistency, since all essential characteristics of an entity are defined once and are inherited when necessary.

3.2.3. Redefinition and inhibition constraints

A. Definitions

The downward default inheritance rule which applies in data models can be precised [Coll 88] [Hamm 81] [LePe 88] [Lenz 85]: one may define constraints which play a role of 'filter' as they precise how the downward inheritance operator applies. However these rules must be in accordance with the *default inheritance* [4] approach:

- one may state that the values of an attribute for the entities of the specific entity type belong to a subdomain of

[4] The description of a specific entity type cannot contradict what is specified in the generic entity type.

the defined attribute domain (this is also worth for attributes of a relationship relation schema);

- one may also restrict the cardinalities i-j of an attribute or of a role, i.e. specify that for the specific entities, these cardinalities are i'-j' such that $i \leq i'$ et $j' \leq j$;
- if an attribute or a role has a cardinality 0-j ($1 \leq j \leq N$), then one may state that the role or the attribute are inhibited by the specific entity type, i.e. entities of that entity type have no value for that attribute or that role.

B. Discussion

Note however that in all cases, these constraints can be avoided by putting the attributes/roles in the specific entity types definition (with perhaps a lack of conciseness).

3.2.4. Multiple inheritance

Multiple inheritance is the mechanism by which entity types in an *is-a* hierarchy are allowed to inherit characteristics from multiple higher level entity types [Peck 88], i.e. multiple inheritance may occur when multiple generalization is allowed. This is convenient for several applications, especially in the context of lifecycle support systems specifications [vLam 90], but it can be difficult to control for both the user and the implementor. Problems arise when a specific entity type inherits a same characteristic from two or more higher level entity types. For example (schema 3.5), a LITERARY_FIGURE might inherit the ADDRESS attribute of PERSON twice, i.e. via AUTHOR and via REVIEWER. And if we specify that this attribute is inhibed by REVIEWER, we have a 'conflict'. Such inheritance conflicts (also naming conflicts) can be resolved by prohibiting multiple generalization or by offering a built-in mechanism for handling conflicts that may arise (for instance, precedence rules) [Borg 88] [Peck 88].

3.2.5. Specialization relationship inheritance

Some relationships between entity types seem to be specialization structures; however an in-depth study shows that they are different [Hain 89d]. For example, we consider the entity type SOLID[5], each occurrence of which represents a geometric solid. It has a name and the formula to compute its volume. We also consider the entity type

[5] Unfortunately, we have not been able to find an adequate example in our library UoD.

MECHANICAL_PART which represents elementary mechanical parts. We have that each mechanical part is a solid, and therefore inherits the name and the formula of the solid. However it is not a generalization/specialization structure as defined previously; indeed several distinct mechanical parts may be the same solid: the domain of MECHANICAL_PART is not included in the domain of SOLID. This relationship is an autonomous concept: it possesses an inheritance mechanism and represents some kind of 'specialization' concept. It seems similar to the *is-instance-of* relation (as a consequence, types and meta-types live in the same schema). Such cases appear in CAD databases and in decision support knowledge bases.

3.3. Class constraints

In order to capture more semantics about objects of the UoD, different constraints concerning the domain inclusion aspect of *is-a* relations can be specified.

3.3.1. Disjunction constraint

A. Definition

Two or more specific entity types having a common generic entity type are *disjoint* if their entity domains are disjoint. Assuming that ES_1, ES_2, ..., ES_n, $n \geq 2$, are (direct or indirect) specific entity types of entity type EG, a subset ES_{i1}, ES_{i2}, ..., ES_{ik} ($1 \leq i_1 < i_2 < \ldots < i_k \leq n$) of these specific entity types form a *disjunction* if for each i and $j \in \{i_1, i_2, \ldots, i_k\}$, $i \neq j$: $ES_i \cap ES_j = \{\}$.

B. Example

Books, journal papers and conference papers are different. Therefore, we can specify a disjunction constraint between them (cf. schema 3.6). This constraint does not stand between DB_BOOK and AI_BOOK which are specific entity types of BOOK, as some books are both database and artificial intelligence books.

C. Discussion

This definition is not restrictive at all, and is too general for being of practical interest. If it is not carefully used, it might lead to inconsistent specifications (for instance, if E_1 *is-a* E_2 *is-a* E_3, the specification that E_1 and E_2 form a disjunction is contradictory with the

definition of the *is-a* relation between E_1 and E_2). Therefore, in order to reduce the complexity of specification management, simplification rules have generally been defined. Frequently, a disjunction constraint is always between direct specific entity types of a common generic entity type. Another rule enforces the definition of maximal groups of disjoint specific entity types in order to avoid redundant information [Coll 88]: ES_{i1}, ES_{i2}, ..., ES_{ik} form a *maximal disjunction group* if there is no other specified group of disjoint specific entity types which is included in the first group (two maximal groups may overlap). One may be more restrictive (and simplify the reasoning) by imposing that all (direct) specific entity types are disjoint, or none of them.

An entity type cannot be a common specific entity type of two entity types forming a disjunction, otherwise its domain is always empty. Therefore, a sufficient condition for single generalization is that all (direct) specific entity types of a common generic entity type are disjoint (but this condition is too restrictive with regard to UoD situations) [Coll 88].

3.3.2. Covering constraint

A. Definition

Two or more specific entity types *cover* a common generic entity type if the domain of the latter is the union of the specific entity types domain. Assuming that ES_1, ES_2, ..., ES_n, $n \geq 1$, are (direct or indirect) specific entity types of entity type EG, a subset ES_{i1}, ES_{i2}, ..., ES_{ik} ($1 \leq i_1 < i_2 < \ldots < i_k \leq n$) of these specific entity types form a *cover* if

$$EG = \bigcup_{i=i_1}^{i_k} ES_i$$

B. Example

We may assume that the books, journal papers and conference papers are the only kinds of publication in the library. Therefore BOOK, JOURNAL_PAPER and CONFERENCE_PAPER form a cover of PUBLICATION (schema 3.7). But AUTHOR does not cover PERSON, since there are many persons who do not write publications.

C. Discussion

As for the previous constraint, we presented an overall definition. For practical reason it is careful to restrict it to direct specific entity

types. By analogy with the disjunction constraint, some authors [Coll 88] propose to eliminate redundant, specifications by enforcing the specification of minimal covering groups only: ES_{i1}, ES_{i2}, ..., ES_{ik} form a *minimal covering group* if there is not any other specified covering constraint between a group of specific entity types which includes the first group (two minimal groups may overlap). Again, a more restricting rule allows only the specification of a covering constraint either on all specific entity types, or none of them.

Note that a single specific entity type may cover its generic entity type: it is a particular case where the domains are always equal but the *is-a* relation has been specified to stress an abstraction (i.e. the *more concrete* relation).

D. Definition

There is a constraint we did not encounter in the literature, but it might be of interest: the cover of the intersection of two or more generic entity types by a common specific entity type. Such a constraint states that the intersection of the generic entity types domains is always equal to the domain of that common specific entity type. Of course, this constraint has a meaning only in case of multiple generalization.

E. Example

We have already seen that a LITERARY_FIGURE *is-a* AUTHOR and LITERARY_FIGURE *is-a* REVIEWER. To represent that only literary figures may be both authors and reviewers, we can specify that it covers their intersection (see schema 3.8).

3.3.3. Partition constraint

A. Definition

A set of specific entity types which are both a disjoint and cover a generic entity type form a *partition* of that generic entity type.

B. Example

If we combine schema 3.6 and 3.7, BOOK, JOURNAL_PAPER, and CONFERENCE_PAPER form a partition of PUBLICATION.

3.4. The generalization/specialization criterion

Often a textual *description* (or *definition*) describes the meaning and contents of classes (with reference to the UoD); this class description should be used to describe the specific nature of the entities that constitute the class and to indicate their significance, their 'role' in the UoD [Hamm 81] [Boda 89]. Therefore it is also interesting to associate to *is-a* relations the viewpoint or criterion used to generalize or specialize, as during the abstraction phase it is often quoted.

3.4.1. Semantic criterion

When we apply the generalization mechanism, we do it according to a point of view, a criterion (which is a reference to the UoD). We call this the *upward criterion* or *generalization criterion*.

On the other hand, when specializing we also use a criterion: it specifies how entities of the generic entity type are 'distributed' as entities of the specific entity types. This criterion, i.e. the *downward criterion* or *specialization criterion*, can be a 'direct' reference to the UoD (we say that it is *manual* or *user-defined*); but it can also be a reference to other information in the database. So we shall see that they are two fundamental uses of 'subtyping' in semantic models:

- to form *user-defined* specific types;
- to form *derived* specific types (the contents of such a specific type can be derived from data stored elsewhere in the schema, along with the definition of predicates).

3.4.2. Database-defined specialization

A. Definition

A specific entity type may be defined as subset of the generic entity type satisfying a given selection criterion, called *specialization criterion* or *assignment criterion* [Afsa 86] [King 81] [Peck 88]. These are *database-defined* specific entity types. Under this aspect, the concept of specific entity type is not far from that of view [LePe 88].

The criterion may be defined either upon a value taken by attributes of the generic entity type (proper or inherited), or on the relationships

that occurrences of the generic entity type have with occurrences of other entity types. Specific entity types are then defined using predicates.

B. Example

In our reference UoD, we can define the BEST_SELLER entity type as a specific entity type of BOOK, and specify that occurrences of the former type are the books with SALES ≥ 10,000 (schema 3.9). It is also specified that database books are books the topic of which is *DB* while artificial intelligence books have *AI* as topic. Incidentally, TOPIC is an inherited attribute from PUBLICATION by BOOK.

C. Discussion

More simply, only one attribute is often declared; its domain includes the values representing the specific entity types. This type of attribute, by nature, need not be inherited to the specific entity types [Saka 83].

This approach is really towards type construction: new types are constructed from other types by using attribute/relationship restrictions on previously defined types. More precisely, given a base class, non base classes are defined. In SDM [Hamm 81] for instance, a specific entity type ES is defined by specifying a generic entity type EG and a predicate P on the occurrences of EG; ES consists of just those occurrences of EG that satisfy P. Predicate P can be defined on attributes of EG, indicating which occurrences of EG are occurrences of ES. The usual comparison operators and boolean connectivities are allowed.

Such derived data, in order to be supported, need the definition of a language for specifying derivation rules. Often [Hamm 81], this language is a variant of the first-order predicate calculus, extended to permit the direct use of attributes names and set operators. In our GER framework, these derivation rules can be expressed by algebraic operator (for instance, the projection).

We may note that this predicate and class constraints are mutually 'linked'. In [Schie 83], the disjunction and the covering constraints are defined with regard to these predicates.

3.4.3. User-defined specialization

A. Definition

We can define a specific entity type ES as a *user-defined* (or *user-controllable*) specific entity type of EG. This means that ES contains at all times only occurrences that are belonging to EG, but the definition of its specific nature does not identify which occurrences of EG are in ES; database users will have to add 'manually' to and delete from ES (so long as the specific entity type limitation is observed).

B. Example

In the library example, good books are identified by the end-user. Therefore entity type GOOD_BOOK can be defined as a specific entity type of BOOK which is user-controllable (i.e. no predicate is associated).

C. Discussion

The difference between a database-defined specific entity type and a user-defined specific entity type is that the belonging to the former is determined by other information in the database, while the membership to the latter is directly and explicitely controlled by users.

Note that it is possible to simulate the effect of a user-defined specific entity type by an database-defined specific entity type. But this would be a confusing and indirect method of capturing the semantics of the UoD. In particular, there are cases where the method of determining specific entity type membership is beyond the scope of the database schema (e.g. by virtue of being complex or because there is no such rule in the UoD) [Hamm 81].

3.5. *Is-a* constructs

After having analysed various facets of *is-a* relations, let us now see how different authors introduced generalization/specialization as constructs of their models.

3.5.1. The problem of defining a generalization/specialization construct

The problem of the introduction of a generalization/specialization construct in a model must typically try to reach a compromise between two dangers [LePe 88]:

- to apply strict rules compel the designer to ask himself questions, and to be more rigorous in his study, but if these rules are too rigid, he will not be able to describe all the real world complexity;
- the absence of any construction rule may have as a result that he builds hierarchies of *is-a* relations which are redundant, unreadable, inconsistent and even difficult to implement; semantics is perhaps easier to model (for an expert designer, at least).

In several models, generalization/specialization is introduced with 'simple' *is-a* relations [Spac 89] [SuYS 86]. At a first sight it seems to be the simplest solution, however if one wants to introduce disjunction or covering constraints, it may introduce complexity: it is difficult to manage the consistency of models where such constraints can appear between each pair of specific (or generic) entity types. And it may be interesting to group specific entity types which correspond to a same generalization/specialization criterion. Therefore some authors proposed 'elaborated' constructs which incorporate in a more or less restrictive and consistent way the different generalization/specialization structures. In spite of the lack of generality, these constructs however are more simple to manage and to understand.

Historically, semantic models have used a single kind of *is-a* constructs for both generalization and specialization purposes [Hull 88]. While many authors are more or less unaware of whether their construct is for generalization or for specialization, more recent research however provides constructs that favour the specification of *is-a* hierarchies in a bottom-up fashion (i.e. the specific entity types are first described and the definition of generic entity types follows) or in a top-down way (i.e. the definition of specific entity types is based on their generic entity type; specialization is used for top-down decomposition, where the most general concepts are first recognized and their specific types are incrementally designed [Qian 85]). As noted in [Davi 89], the implied semantics

of *is-a* relations in such models, where it seems depicted in only a single direction, may lead to forget the specification of important constraints.

Some proposals have differentiated several kinds of *is-a* constructs, and some incorporate more than one construct in the same model. The other motivations for distinguishing kinds of *is-a* constructs stem from studies on their update semantics, and from works on schema integration [Hull 87b]. Note that having too many concepts, may introduce a 'choice problem'.

3.5.2. Different proposals

To give the reader an idea of some of these 'elaborated' constructs, we shall now explain, with our vocabulary, several of them we consider being 'typical'. In chapter 6, we shall also propose a concept.

A. The cluster of Smith and Smith [Smit 77]

A cluster is defined as a group of specific entity types; it has a name and a criterion. It is an intermediate concept between the generic entity type and the specific entity types.

B. The category of Sakai [Saka 83]

A category functions as a node to separate entity types into different levels of abstractions making up a hierarchy of different views of objects. A derived attribute may be defined.

C. Three types of generalization/specialization constructs [Czed 90]

The first type involves exactly two entity types: the specific and the generic entity type. The second type involves one generic entity type and any number of specific entity types forming a covering. The third type also involves a generic entity type and any number of entity types, but they are forming a partition.

D. The cluster of Davis and Bonnel [Davi 89]

Davis and Bonnel allow one to use single *is-a* relations and to define class constraints with cardinalities and inter-types constraints, but they also define the concept of cluster which gives the ability to define multiple hierarchies (which may intersect). To their opinion, having multiple taxonomies provides some level of semantic relativism, allowing

for some degree of subjectivity in how concepts in the schema are structured.

E. The subset hierarchy and the generalization hierarchy [Teor 86]

Teorey et al. make the distinction between specialization and generalization:

- a subset hierarchy is defined between two entity types ES and EG; it represents the *is-a* relation;
- a generalization hierarchy is formed by a generic entity type and a set of specific entity types which form a partition.

F. The subset hierarchy and the exclusive (complete) generalization hierarchy [Berg 88]

The subset hierarchy is the same as Teorey's. A generic entity type is an exclusive generalization of its specific entity types if the latter are disjoint. The exclusive generalization is complete if it forms a partition.

G. The *is-a* interconnection in SDM [Hamm 81]

Hammer and Mc Leod typically emphasize top-down design of types. A specific entity type is always defined in terms of another entity type: we have first to define a generic entity type, and then specify the specific entity types by 'restriction', i.e. using a predicate. The latter may be user-controllable or database-defined.

3.6. Another generalization/specialization relation: the *may-be-a* relation

Relations for generalization/specialization have also been proposed which do not involve all occurrences of the specific entity type.

3.6.1. The *may-be-a* relation

The *is-a* relation and its inclusion dependency are the cornerstone of the generalization/specialization constructs in data models. But, as quoted in [Spac 89], it has been largely advocated for exceptions to that fundamental class inclusion property; the famous *is-a* relation should sometimes

be named *99% is-a* relation. This relation has been called *may-be-a*. We say that an entity type ES *may-be-a* entity type EG if ES ∩ EG <> {} is possible and if ES is *more concrete* than EG.

For example, SHAREHOLDER *may-be-a* EMPLOYEE means that some shareholders may be employees of the library.

The *is-a* relation is a special case of the *may-be-a* relation (ES is-a EG => ES may-be-a EG).

An inheritance mechanism is also associated with that relation.

We may note that often, if not always (with regard to the examples used in the different descriptions of these concepts), the *may-be-a* relation hides a multidomain role[6], i.e. this relation holds between entity types because of their participation to any role, and is not based on the intrinsic nature of the entities.

Moreover, a *may-be-a* relation can always be represented by an *is-a* relation (if multiple generalization is allowed) [Coll 88]:

> ES *may-be-a* EG <=> ES *is-a* E_1, ES *is-a* E_2, E_1 *is-a* EG (where E_1 contains entities belonging to ES ∩ EG, and E_2 contains the other entities).

3.6.2. Class constraints

We can define class constraints for the *may-be-a* relation. If we have that ES *may-be-a* EG_1, ES *may-be-a* EG_2, ..., ES *may-be-a* EG_n, $n \geq 1$, we can specify that:

- EG_{j1}, EG_{j2}, ..., EG_{jk} $(1 \leq j_1 < \ldots < j_k \leq n)$ are *disjoint*, i.e. that for each i and $j \in \{j_1, j_2, \ldots, j_k\}$, $i <> j$: $EG_i \cap EG_j = \emptyset$;
- ES is *included in the union of* EG_{j1}, ... EG_{jk} $(1 \leq j_1 < \ldots < j_k \leq n)$, i.e. that

$$ES \subseteq \bigcup_{j=j1}^{jk} EG_j .$$

[6] A multidomain role is defined on the union of entity types instead of on one entity type [Hain 89d].

3.6.3. The *may-be-a* constructs

Generally, this relation has been introduced in a complex construct; Spaccapietra et al. [Spac 89] propose however both 'basic' *is-a* and *may-be-a* relations in their model.

A. The alternative generalization [Codd 79]

Codd specifies that the usual concept of generalization hierarchy may be increased by noting that an entity type may be generalized into two or more alternative types, i.e. an entity of the first type belongs to any entity type among these alternative types.

B. The category [Elma 85]

The category construct is a complex, hybrid concept [Coll 88] which can have different objectives. Thus a category C is defined as follows:

$$C = E_1\ [P_1] \cup E_2\ [P_2] \cup \ldots \cup E_n\ [P_n],$$

where P_i, $1 \leq i \leq n$, is an optional predicate restricting E_i domain, i.e. specifying exactly those entities of E_i that are members of the category C.

It allows the representation of:

- the 'traditional' inclusion dependency of *is-a* relation (ES = EG [P]);
- the cover of the generic entity type by the specific entity types ($EG = ES_1 \cup ES_2 \cup \ldots \cup ES_n$);
- Codd's alternative generalization ($E = ES_1\ [P_1] \cup \ldots \cup ES_n\ [P_n]$).

C. Simple generalization, alternative generalization, multiple generalization, selective generalization [Roch 88]

Rochefeld and Morejon propose the integration of four generalizations constructs in their extension of the Merise conceptual data model:

- the *simple generalization* allows one to define *is-a* relations;
- the *multiple generalization* is used when a specific entity type inherits from several types;

- the *alternative generalization* represents the fact that an entity type inherits exclusively from one of other entity types;
- the *selective generalization* corresponds to Codd's alternative generalization.

3.7. Generalization/specialization for relationship types and attributes

The generalization/specialization structures as they have been defined could be adapted for relationship types and attributes. The interested reader may consult [Coll 88], where an in-depth analysis is proposed. Note that only a few authors propose these concepts.

To our mind, it is better to avoid the introduction of generalization/specialization in this 'level' in order not to complexify the model. If *is-a* relations seem to be useful for attribute or relationship types, one may then 'promote' them to entity types.

Chapter 4

Generalization/specialization and database design activities

After having presented the concepts of generalization/specialization, we shall explain the 'interaction' between different design activities in the conceptual level and generalization/specialization constructs.

In chapter 2, we have seen that the concerned activities involve:

- the design of a subschema;
- the integration of subschemata;
- the mapping between schemata;
- the restructuration of a schema according to a user viewpoint.

We have also seen that several rules and transformations are at the very basis of these activities.

In this chapter, we shall see:

- the rules or guidelines for the construction of a schema with generalization/specialization constructs;
- transformations which concern generalization/specialization.

4.1. Introduction of generalization/specialization constructs in schemata

Informally speaking, *is-a* relations may be used in a conceptual schema for two reasons [Hull 87b]:

- to represent one or more possible overlapping specific entity types of an entity type;
- to form an entity type that contains the union (or is covered by the union) of entity types already in the schema.

Indeed, one may distinguish two approaches for the building of schemata which include generalization/specialization constructs (of course, one mix them according to circumstances):

- the first approach consists of course in introducing *is-a* relations 'on-line' when they are obvious in the UoD;
- the second approach lies under 'a posteriori' introduction of generalization/specialization constructs (it consists therefore in some kind of reorganization of schemata).

The second approach, especially, is part of the view intregation process, but is also proposed in 'single' schema methodologies: in [Ceri 81] and [Teor 86], entity types are first designed and then generalization/specialization hierarchies are introduced.

But what are the guiding 'events' for detecting the need of generalization/specialization structures? Since the *is-a* relation includes two aspects, and is characterized by an inheritance mechanism, these 'events' may concern:

- inclusion dependencies or intersections between entity types domains;
- descriptive aspect of the types.

For each pair of entity type, the designer can supply information about these aspects. In the first approach, these aspects are taken directly. The second approach allows to eliminate imperfections of the data model (for instance, a relationship type of name IS_A may hide an *is-a* relation).

4.1.1. Intersection between entity domains

One may detect four kinds of intersection information between entity types domains:

- the two entity domains are the same;
- one is contained in the other;
- they are overlapping;
- they are disjoint.

Note that the following considerations are valuable for schema design, and some are more intended for schema integration.

A. Identical entity domains

In this case, one entity type may be designed only, except if they are representing two levels of abstraction in the UoD.

B. Inclusion of a domain into another domain

Then one can define an *is-a* relation from the included entity type towards the other, if it means that one is more concrete than the other. Attributes and relationship types are reorganized.

C. Two or more domains are overlapping

In that case, one cannot introduce *is-a* relations between them. However it is perhaps worth specifying a common generic entity type, grouping common characteristics, while the overlapping entity types keep their own.

D. Two or more disjoint domains

It is perhaps possible to introduce them as disjoint specific entity types of a newly created entity type. As for the previous case, one may think about any covering constraint too.

4.1.2. Descriptive considerations

The analyzis of characteristics of entity types may allow one to reorganize them, and create *is-a* relations.

When an entity type has attributes with minimum cardinality zero or if it plays a role the cardinality of which is zero and if these zeros mean that a value for these attributes/roles is not allowed for some occurrences, then one may introduce two entity types. Another case may happen when a 'type' attribute appears, or when an exclusion/inclusion/equality constraint exists between roles/attributes. In such cases, where only a few values are allowed, these attributes/roles determine the existence of other attributes/roles. A study of identifier constraints may be useful too.

4.1.3. Global considerations

One may, after the analysis, either create new generic entity type, or create new specific entity types. However these additions are worth only if they are significant in the UoD, i.e. if they bring semantic clarifica-

tion. One thing one must be cautious of, is not to build networks of *is-a* relations which might confuse the reader. To our mind, *is-a* relations should concern the proper nature of objects and not be concerned by their role in relationships. The latter cases can usually be represented by multidomain roles.

Depending on the model used, the consistency between constraints is more or less difficult to verify.

One may also remark that *is-a* relations can replace reflexive relationship types (allowing them to capture more semantics) [Hain 89d] [Roch 88]. For example, if we are interested in recording the husband or the wife of the persons recorded in the library database, a solution is to define a reflexive relationship type attached to PERSON (schema 4.1); however this can also be expressed more precisely by *is-a* relations as proposed in schema 4.2.

In summary, as remarked in [LePe 88], the concept of generalization/specialization in a model allows one:

- to eliminate non significant, null values (and so, enhance the comprehensiveness of a schema);
- to describe in a more precise way the different objects, while keeping a schema easy to understand;
- to model more contraints concerning relations between objects.

4.2. Transformations concerning generalization/specialization constructs

In chapter 2, we saw that transformations are useful tools in current database design methodologies. Concerning the generalization/specialization structures, we find in the literature different transformations to replace them. One approach consists in the definition of mappings with constructs of the implementation model, e.g. with relational constructs [Teor 86] [Smit 77] [Elma 85] [Berm 86] ([Kung 89] proposes to replace them by views) or with CODASYL constructs [Elma 85] [Berm 86]. In another approach, generalization/specialization constructs of an E-R schema are mapped to basic E-R constructs [Berg 88] [Coll 88] [Hain 89d] [Spac 89] [Vigi 90]. Other transformations may be useful during the elaboration of a conceptual

schema; however, we found only one paper [Mann 88] dealing with them, with view integration concerns.

We shall explain the general principle of transformations replacing generalization/specialization structures by basic GER constructs (in [Coll 88] they are studied for an E-R model). It is indeed too complex to detail a transformation outside the context of a particular construct. In chapter 6, we properly define a set of transformations concerning the generalization/specialization construct we propose and also transformations available for the design of schemata.

We assume that these transformations apply on a schema which is correct.

4.2.1. Elementary transformations

Three possible transformations are proposed: the materialization of *is-a* relation by relationship types, the representation of the sole generic entity type, and the representation of the sole specific entity types. The transformations have to preserve the fact that entity types populations 'intersect', and also that entity types share some characteristics.

The simplest solution consists in performing these transformations on a generic entity type and its (direct) specific entity type. However, as class inclusion constraints can be specified between different specific entity types, it does not work properly. It seems then better to consider a generic entity type and all its (direct) specific entity types. In [Mark 89] the representation of multiple generalization by relational constructs is examined: he considers a specific entity type and its generic entity types.

The related integrity constraints (redefinition, inhibition, etc.) have to be transformed too. We did not find any paper dealing with them in a 'general way', due certainly to the complexity and the number of possible combinations.

A. Representation of *is-a* relations by relationship types

This transformation is frequently proposed and seems to be the most natural one. An *is-a* relation is replaced by a 'special' relationship type, the aim of which is to put in correspondence entities representing a same object of the UoD. The specific entity type plays a role of cardinality

1-1[1], while the generic entity type role is of cardinality 0-1. Disjunction constraints are represented by an integrity constraint specifying that an entity of the generic entity type can be associated with at most one of the concerned specific entity types (exclusion of roles). Covering constraints are represented by an integrity constraint specifying that each entity of the generic must be associated with at least one specific entity (existence of roles)[2].

Schema 4.4 shows the application of that transformation to schema 4.3 (previous page).

This transformation however introduces complexity in the management, since an UoD object is represented in different places in the database.

B. Representation of the sole generic entity type

This transformation keeps only the generic entity type to which attributes and roles of the specific entity types have been added (upward inheritance). Attributes of the specific entity types are grouped in an optional, compound attribute. Each inherited role becomes optional, too. A new attribute is added to the generic entity type; it allows to detect the specific 'aspect' of each entity. It is a 'meta-attribute'; its domain of values is the set of the names of the specific entity types. If the specific entity types form a partition, then this attribute is simple and mandatory. If they do not cover the generic entity type, then it is optional. If they are not disjoint, then it is repetitive. Note that equivalence constraints between values of that attributes and the inherited attributes/roles must be specified, as shown in schema 4.5 (next page) which results from the application of that transformation on schema 4.3.

This transformation presents a unique representation of the UoD objects, however there are many integrity constraints which must be specified (note also that a key of a specific entity type becomes a partial key[3] in the generic entity type).

[1] If we were replacing a *may-be-a* relation, that cardinality would be 0-1.
[2] One could use the multidomain role construct.
[3] A *partial key* plays the role of key only for a subset of the instances of the entity type.

C. Representation of the sole specific entity types

This transformation eliminates *is-a* relations by representing only the specific entity types, after having applied downward inheritance of the attributes/roles of the generic entity type. Each attribute of the generic entity type is put in each entity type. The roles are replaced by multi-domains roles[4]. Each key of the generic entity type becomes a global key. This transformation is perfect for partitions but if the specific entity types do not cover the generic entity type, a new entity type must be added to represent entities which do not belong to any specific entity type. If the specific entity types are not disjoint, then three solutions can be proposed to keep the idea that an UoD object may be 'seen through' several specific entity types [Coll 88]:

- if the generic entity type has a key, then that key can be stated as 'reference key', i.e. it is a key declared on several entity types which plays the normal role of key, but if two entities of two different entity types have the same value for the key, this means that they represent the same object of the UoD;
- another solution consists in defining a binary relationship type for each possible intersection between two non disjoint specific entity types; each entity type plays a role of cardinality 0-1, and two entities are linked if they represent a same object of the UoD (note that if we have n, non disjoint specific entity types, $n \geq 2$, we shall need C_n^2 relationship types).
- a final solution consists in producing new specific entity types representing each possible intersection (2 by 2, 3 by 3, ..., n by n) between n specific entity types, $n \geq 2$, (we need $\sum_{j=2}^{n} C_n^j$ new entity types).

The result of the application of this transformation on schema 4.3 is shown in schema 4.6.

[4] If multidomain roles are not allowed, we can represent them by 'normal' roles, as explained in the footnote 8 of chapter 6 and presented in the following example (these transformations are also proposed in [Vigi 90], for instance).

This transformation respects the principle of single recording of UoD objects, but needs numerous integrity constraints as the precedent technique.

4.2.2. Complex transformations

After the analysis of elementary transformations, one must consider how they can be combined to eliminate *is-a* relations from a schema. The reader may refer to [Coll 88], and to chapter 6 for our generalization/specialization construct.

Chapter 5

The TRAMIS database design workbench

In this chapter, we forget for a while the generalization/specialization structures and enter the world of computer-aided software engineering. We present TRAMIS, a database design workbench developed by the Institut d'Informatique and marketed by Concis. We first describe its architecture and then explain its model: an E-R model. Afterwards we overview its processors and the user-interface principle. For all these points we provide two levels of explanation:

- a 'user-oriented' description stressing the aspects relevant to TRAMIS users;
- a technical description interesting for TRAMIS implementors.

In the latter type of descriptions, we try to provide an explanation being precise enough for a 'direct' implementation, while leaving aside details which might confuse too much and be impediment to the generality of the text.

5.1. TRAMIS architecture

5.1.1. User level explanations

TRAMIS is an autonomous environment for the design of information systems; this software tool helps the designer(s) of an information system to analyse, design, and produce a correct, operational, and efficient database.

TRAMIS is divided into two communicating components (see figure 5.1):

- TRAMIS/VIEW, a graphical specification tool for the design of E-R conceptual schemata;
- TRAMIS/MASTER, the workbench providing the database designer with assistance during all steps of the design of database.

In the following, we focus only on TRAMIS/MASTER (henceforth, called TRAMIS). A complete description may be found in [Conc 90a] and [Conc 90b].

The main methodological characteristics of TRAMIS are:

- a unique E-R model for conceptual, logical, physical and DBMS schemata;
- a design process based on manual, aided or automated schema transformations;
- a forward engineering tool with statistical inference;
- permissive schemata elaboration with validations;
- the building by the user of its own models according to its customized methodological approach.

The schema of the database under project is stored in the specification database. This database is organised for the recording of conceptual, logical, and physical E-R schemata. TRAMIS has been designed under the *toolkit* paradigm, following the *database* approach: functions of the workbench are performed by independent processors, working on the contents of the specification database. Moreover, functions may be applied in any order: in this sense, the tool is somewhat independent on any methodological coloration. As a consequence, functional extensibility is easier: adding a new function has no impact on the specification database, nor on the existing functions. Adding a new concept, on the other hand, implies a modification of the specification database and an update of the consequent functions (this will be our job in next chapter).

The user interface is based on the *Windows* approach. According to the object-oriented user interface, the user selects an object; and he chooses the function to be performed on that object (via a menu). Dialogues are then the interaction point between the user and the tool.

5.1.2. Technical level explanations

TRAMIS is running on personal computers in a Windows environment. It has been implemented in C [Kern 84] [Plum 86] with the Microsoft Windows toolkit [Petz 88] [Sacr 88]. The specification database is a CODASYL database [Hain 85].

5.2. TRAMIS model

5.2.1. User level explanations

TRAMIS is based on a single E-R model for all steps of database design. Its constructs[1] are those of current E-R models (schema, entity type, relationship type, attribute, identifier group), adapted with aspects for logical and physical design (access key, reference group, space). Because of their usefulness in a workbench, other constructs have been added: static and dynamic quantifications, textual description and technical note, origin.

An approach offering a single model for defining conceptual schemata, schemata with logical accesses, a relational or CODASYL optimized schema for instance, offers the power of multi-models methods, while keeping the simplicity and uniformity of mono-model approaches [Conc 90a].

As we are essentially concerned with conceptual aspects (according to current methodologies), we shall not consider the other constructs (an interested reader may consult [Conc 90a] for a full explanation).

We do not claim to provide here an introduction nor a complete presentation of the E-R approach; for such the reader may consult other references [Boda 89] [Chen 76] [Dols 88] [Vigi 90].

The current TRAMIS conceptual model (called the *basic* model) has the following concepts: schema, entity type, relationship type and role, attribute, identifier group, description, origin technical note, and static statistics.

A. Schema

A *schema* describes how constructs are assembled. It contains mainly entity types, relationship types and attributes. It has a *name* and a *date*.

B. Entity type

An *entity type* represents a class of objects of the UoD[2], perceived as having their own existence. It is worth mentioning here that the classi-

[1] Conventionally, we say that a model has constructs which may be assembled according to some construction rules. As a database design workbench rests under (a) model(s), we shall speak of the constructs of the workbench, referring to the constructs of the model(s) it supports.

[2] We distinguish between the object of the real world and the representation in the database.

fication abstraction mechanism is at the basis of that definition; an entity type encompasses both a class aspect and a type (descriptive) aspect.

Each entity type has a *name*[3] and a short name identifying it within a conceptual schema, and is characterized by a *date*. An entity type is graphically represented by a rectangle, the head of which contains the name of the entity type. For example figure 5.2 contains two entity types: PUBLICATION and AUTHOR.

The *population* of an entity type is the set of entities (or occurences) belonging to that entity type at a given time. In the basic model, populations of entity types are always disjoint, i.e. an entity belongs to one and only one entity type. It is also worth mentioning that entities are distinct and exist by themselves.

C. Relationship type

A *relationship type* represents a class of links between objects of the reality. Each relationship type is characterized by a *date*, identified within a conceptual schema by its *name* and its shortname.

A relationship type is defined between entity types (representing the class of involved objects); each entity type plays a *role* in the relationship type (this means that each relationship is a group of two or more entities, each one playing a role in this group). At least two roles are defined for each relationship type (the number of roles is called the *degree* of the relationship type). An entity type may play many roles in a single relationship type. Each role has a *name* which identifies it within those of the relationship type (but it is allowed to give it the name of the entity type). A relationship type is graphically represented by an hexagon with a heading containing its name and linked to the participating entity types by arcs labelled by the name of their roles, except if it is the name of the entity type, in which case the name remains implicit. For example (figure 5.3), relationship type WRITING between AUTHOR and PUBLICATION represents that authors write publications.

The *population* of a relationship type is the set of relationships belonging to that relationship type at a given time. Relationships belong to one relationship type in the basic model.

[3] In the following, we shall always denote constructs by their name.

Each role is characterized by a *cardinality constraint*. This is a couple of integers i-j, where i ≥ 0 , j ≥ 1, j ≥ i, specifying that each entity plays from i to j times this role; i and j are respectively called the *minimum cardinality* and the *maximum cardinality*. The usual values for i are 0 (in which case the role is said *optional*) and 1 (for a *mandatory* role). When j denotes a number arbitrarily great, it is represented by letter *N*. In the diagrams, the cardinality is put on the line of the role (cf. figure 5.3).

D. Attribute

An *attribute* is the representation of a property common to the objects of the UoD corresponding to an entity type, or to the links corresponding to a relationship type; for each instance of the entity type or of the relationship type, it can take 0, 1 or many values (not necessarily distinct).

It is worth mentioning that an entity type or a relationship type may have no attribute.

An attribute is associated with a *cardinality constraint*, i.e. a couple of integers i-j (satisfying i ≥ 0, j ≥ 1, j ≥ i) specifying that the number of values associated to each occurrence is between i and j. When i = 0, the attribute is said *optional*; otherwise it is *mandatory*. When j = 1, the attribute is *simple*; otherwise it is *repetitive* (its repetitivity is *fixed* if i = j, and *variable* else).

An attribute may be constituted with component attributes. Such an attribute is said *compound*. Otherwise, it is *elementary*. The components of an attribute themselves may be compound or elementary. An attribute which directly belongs to the entity type or relationship type is said of *upper level*. An elementary attribute has a *domain of values*: it is the set of possible values for that attribute; these values obey the same laws of manipulation and have the same structure. They are characterized *by their type* (or *format*) and eventually their *length* and *number of decimals*. In TRAMIS, domain values are restricted to the following types: alphanumeric, numeric, date, boolean.

Each attribute has a *name* identifying it among attribute of the relationship type or the entity type (for attributes of upper level) and between the other components of a compound attribute. Graphically, we put

the name of the attributes in the lower part of the entity type or relationship type graphics. Optional attributes are within parentheses and their cardinality is put between brackets. The structure of a compound attribute is indicated by an indentation at each level of decomposition. Figure 5.4 presents the attributes of PERSON.

E. Identifier group

The idea of the identifier constraint[4] is that a certain amount of information allows one to 'practically' identify an occurrence of an entity type or relationship type among other occurrences of the population.

An *identifier group* of an entity type (or relationship type) is a group of attributes and roles so that, given a value for these attributes/roles, there cannot exist more than one occurrence with this value. Note that such a constraint is not necessarily specified.

The identifier constraint is often graphically represented with an underlined notation. For example, the entity type PUBLICATION is identified by its ISBN_CODE (figure 5.5).

F. Textual descriptions

A *textual description* may be associated with each construct of the schema. It is a free text introduced and modified by the designer. It is aimed at describing, in natural language, the semantics of the described construct. The text follows the construct during the design of the schema, whatever transformations are performed on it.

G. Technical note

The *technical* note is a free text which may be associated with each construct of TRAMIS. Both TRAMIS and the designer inscribe information about transformations performed, as well as their technical reasons. They are intended to the programmer and for further maintenance of the database.

H. Origin

When a conceptual construct is transformed, it is important to keep trace of its origin. TRAMIS indicates by the *origin* property of a construct the name of the construct it comes from, if any.

[4] This is the only integrity constraint, with cardinality, which is proposed to the user in version 1.01 of TRAMIS conceptual model (the inclusion constraint is available in the latest version).

I. Statistical aspects

The *static quantifications* description give statistical information about the size of entity type or relationship type populations, and about the length and frequency of attributes. This information can generally be derived from observation of the UoD, and so recorded in the conceptual schema.

a) Population average size of an entity type

The *population average size* N_E of the entity type E population is the average number of occurrences of that entity type in the database in a reference instant.

b) Average cardinality of a role

The *average cardinality* μ_r of a role r of a relationship type R is the average number of occurrences of R in which entities play role r.

If cardinality of r is i_r-j_r, then we have $i_r \leq \mu_r \leq j_r$.

c) Population average size of a relationship type

The *population average size* N_R of relationship type R is the average number of relationships of R in the database at a given time. Actually, if r is a role of R played by entity type E, we have: $N_R = \mu_r * N_E$. This property allows one to compute the average cardinality of roles when one of them is known, since for each role r_i (played by E_i) and r_j (played by E_j), we have: $\mu_{rj} = \mu_{ri} * N_{Ei} / N_{Ej}$.

d) Frequency of an attribute

If an attribute A has a cardinality i_A-j_A, where $i_A < j_A$, its *frequency* μ_A is defined as the average number of values associated with each entity, relationship, or compound father attribute. We have: $i_A \leq \mu_A \leq j_A$.

e) Average length of an attribute

For each alphanumeric attribute A, we may mention the *average length* σ_A of its values in the database.

5.2.2. Technical level explanations

The specification database is the heart of TRAMIS. It records information concerning the design of the database at every stage, in terms of the constructs of TRAMIS model.

We shall now present the subschema of this database being interesting for the integration of the G/S structures in TRAMIS. For a complete description, the reader may refer to [Hain 86b].

Actually, the approach which guided the design of the specification database tried to reach a compromise between the generality of the concepts, allowing easier modifications, and precise definitions, offering a detailed and explicit description of a particular schema [Hain 86b]. Here, we describe - avoiding unnecessary details - the particular, restricted specification database schema used in version 1.01 of TRAMIS[5]: it explains the conceptual model of the workbench as far as current methodologies and tools are concerned and present only what is useful for the user.

The schema will be explained using an E-R model[6]. Following the methodology developed in [Hain 86a], the GAM schema will then be derived.

A. The E-R schema of the specification database

The specification database (figure 5.6) contains the description of one - in current version - information system (SYSTEM), identified by its name (SYS_NAME); the description of data is recorded in one - restriction of version 1.01 - schema (SCHEMA) attached to the SYSTEM, via SCH_OF_SYS. In the SYSTEM, the SCHEMA is identified by its name (NAME) and is characterized by a date (DATE).

A SCHEMA may contain entity types (ENTITY_T) and relationship types (LINK). Each ENTITY_T belongs to one schema via E_IN, as well as each LINK via L_IN.

ENTITY_Ts of a SCHEMA are identified by their name (E_NAME) or by their short name (SHORT_NAME). They are characterized by a date (DATE) and the average size of their population (POPULATION)[7].

[5] For an easy reading, we took the liberty to modify certain constants or names.

[6] The E-R model used for the description of this (meta-)schema includes the multidomain role construct [Hain 89d] (showing incidentally its interest).

[7] This is a simplification for conciseness purposes.

LINKs of a SCHEMA are identified by their name (L_NAME) or by their short name (SHORT_NAME). They are characterized by a date (DATE), the average size of their population (POPULATION) and their degree (DEGREE).

ENTITY_Ts participating in a LINK play a role (ROLE) via E_ROLE; a LINK is concerned by at least two ROLEs and at most four ROLEs - current restriction - via L_ROLE (the DEGREE of a LINK equals the number of associated ROLEs). Each LINK belongs to the SCHEMA of the ENTITY_Ts of its ROLEs. ROLEs are played by one ENTITY_T and concern one LINK. ROLEs are characterized by their name (R_NAME), a minimal cardinality (MIN_CON) and a maximal cardinality (MAX_CON); the conventional cardinality *N* will be represented by value *99999*. They have also an average cardinality (AVG_CON) with MIN_CON ≤ AVG_CON ≤ MAX_CON.

An ENTITY_T or a LINK may have, via ATT_OF, attributes (ATTRIBUTE), which may themselves have ATTRIBUTEs. ATTRIBUTEs are characterized by a name (AT_NAME), a format (FORMAT), a maximal length (LENGTH), and by the number of decimals for numerical values (DECIM). The current supported values for FORMAT are *Alphanumeric*, *Numeric*, *Real*, *Date*, *Boolean*, *Compound*. Only ATTRIBUTEs with FORMAT = *Compound* may have ATTRIBUTEs. DECIM takes a value only when FORMAT = *Numeric*. ATTRIBUTEs have a minimal cardinality (MIN_REP) and a maximal cardinality (MAX_REP); an illimited cardinality is represented by the value *99999*. Statistical aspects are their average length (AVG_LENGTH) - 0 ≤ AVG_LENGTH ≤ LENGTH - and their average repetitivity (AVG_REP) - MIN_REP ≤ AVG_REP ≤ MAX_REP. POSITION indicates the position of an ATTRIBUTE in a given level.

Identifier groups of both ENTITY_Ts and LINKs are recorded in the entity type ID_KEY_ORD, where the attribute TYPE is *Id*. They are linked to their ENTITY_T or LINK via IKO_OF. An ID_KEY_ORD is composed of ATTRIBUTEs and/or ROLEs (via COMPONENT). Attribute TYPE_O of COMPONENT indicates (redundantly) if the component is an ATTRIBUTE (value *Attr*) or a ROLE (value *Role*). The STATUS attribute of an ID_KEY_ORD indicates if it is a primary (*Prim*) or a secondary (*Sec*) identifier. Each ENTITY_T or LINK has only one primary identifier group. The position of the components is recorded in SEQ_NBR. An identifier code (IKO_CODE) is associated with each ID_KEY_ORD. Moreover, the following constraints are defined:

- if a COMPONENT of an ID_KEY_ORD of an ENTITY_T or LINK is an ATTRIBUTE, the latter belongs to that ENTITY_T or LINK;

- if a COMPONENT of an ID_KEY_ORD of an ENTITY_T is a ROLE, the latter is played by that ENTITY_T;
- if a COMPONENT of an ID_KEY_ORD of a LINK is a ROLE, the latter belongs to that ENTITY_T.

The entity type PROPERTY is used to record, via PROP_OF, the origin of a conceptual 'construct', i.e. an ENTITY_T, a LINK, or an ATTRIBUTE. The attribute P_ROLE of PROPERTY takes the value *Origin*, the P_VALUE attribute records the name of the origin, and TYPE contains its type.

Finally, almost each conceptual construct (SCHEMA, ENTITY_T, LINK, ATTRIBUTE) may be associated, via DESCR_OF, with a textual description (DESCRIPTION with attribute TYPE = *Desc*) and a technical note (DESCRIPTION with attribute TYPE = *Tech_n*). Each DESCRIPTION is attached to a SYSTEM through DESCR_IN (the DESCRIPTION of a construct must be attached to the SYSTEM of that construct). The corpus of a DESCRIPTION is a free text of any length (TEXT).

B. The GAM schema of the specification database

The GAM schema (figure 5.7) is obtained by application of 'traditional' transformations described in [Hain 86a]. It is worth remembering that figure 5.6 presents only aspects of a particular interpretation of the schema; therefore certain relationship types without attribute in diagram of figure 5.6. have been transformed to record types.

We assume that each link constitutes an access path, that the constraints explained in the previous point have been correctly translated (but space is lacking here to express them), and that attributes are translated into items (they are not drawn on the schema in order not to overload it).

5.3. TRAMIS processors

5.3.1. User level explanations

We have seen that independent tools are working on the specification database for the design of the database.

The main processors of TRAMIS are (figure 5.8, next page):

- management;
- consultation;

Errata

We hope the reader will forgive us the following mistakes we missed in spite of many proof-readings.

The following corrections have an impact on the 'semantics' of the work:

- page 30, line 16: add 'direct' before 'generic';
- page 64, line 22: replace 'repetitivity' by 'cardinality';
- page 78, line 10: replace 'N_{Esi}' by 'N_{Esj}';
- page 79, line 8: replace '$j \geq \Sigma\ j_k$' by '$i \geq \Sigma\ i_k$' and suppress next line;
- page 84, line 30: replace 'type' by 'types';
- page 86, line 13: add ', the same cardinality, the same quantifications, ' after 'components';
- page 88, line 14: replace '$i=1$' by '$i=2$';
- page J, line 6 & page O, line 25: replace 'RELATIONSHIP' by 'ENTITY';
- page M, note 10: replace 'This menu is' by 'The transformations for the elimination of categories are';
- page O, lines 8 & 9: add '/removal' after 'Adding'.

Trying to reach perfection (because 'writing makes an exact man' to Bacon's mind), the following list presents corrections having a 'syntactical' impact, i.e. enhancing the quality of the English of the text:

- page 2, line 13: replace 'database' by 'databases';
- page 3, line 10: replace 'transformation' by 'transformations';
- page 3, line 16: remove the ',';
- page 6, line 13: replace 'classes' by 'class';
- page 10, line 12: replace 'becomed' by 'become';
- page 18, line 15: replace 'ressources' by 'resources';
- page 18, line 27: add a ',' after 'that';
- page 19, line 27: add 'are' after 'They';
- page 28, line 24: replace 'relationship' by 'relationships';
- page 36, line 21: replace 'form' by 'forms';
- page 37, line 21: replace 'domain' by 'domains';
- page 47, line 14: replace 'attribute' by 'attributes';
- page 53, line 23: replace 'attributes' by 'attribute';
- page 60, line 33: replace 'attribute' by 'attributes';
- page 70, line 22: replace 'are' by 'is';
- page 74, line 15: add a ',' after 'type';
- page 76, line 10: add 'by' after 'expressible';
- page 76, line 18: replace 'identifiant' by 'identifier';
- page 80, line 7: replace 'constructs' by 'construct';
- page 85, line 31: add 'with' after 'compliant';
- page 86, line 4: replace 'verified and; statistical' by 'is verified and statistical';
- page 89, line 24: add 'and' after 'type,';
- page 91, line 12: add 'on' after 'applies';
- page 95, line 32: replace 'pratical' by 'practical';
- page A, line 13: replace 'above' by 'hereafter'.

- transformation;
- model compliance checking;
- production of executable descriptions;
- reporting;
- import/export.

A. Management

We may distinguish two levels: the functions manipulating the specification database independently of its contents, and those which create, update and delete the constructs. These operations are validated against a certain degree of integrity (for instance, each relationship type has at least two roles). But some temporary inconsistencies, which are typical to incremental design, are allowed.

B. Consultation

This processor allows the selection of a schema or an object, and the consultation of its characteristics (for an entity type, for instance, we may consult its date, its origin, its name, its short name, its attributes, the roles it plays in relationship types, its groups, its statistical aspects, its semantic description, and its technical note).

C. Transformation

A transformation is an operation which modifies a schema while preserving a certain number of conceptual, statistical and technical aspects. Preservation is guaranteed by the existence of an inverse transformation allowing to retrieve the initial transformed schema. We have seen (chapter 2) that they are at the basis of current design methodologies. They are useful in a workbench for several reasons [Conc 90a]:

- model compliance (for example, obtain a CODASYL schema from an E-R schema);
- production of efficient schemata;
- progressive elaboration of a conceptual schema.

Here follow some of the transformations that have been implemented in TRAMIS and that are of interest as far as conceptual modelling is concerned (other transformations are presented in [Conc 90a]). We may remark that

TRAMIS offers three modes of transformation uses:

- *punctual transformations* which consist in the application of a transformation on one construct (the designer is given entire control and TRAMIS ignore the objective);
- *global transformations* where a transformation is performed on all the constructs on which it can be applied (the designer has less control, although he may refuse a transformation, but the transformation process is faster);
- *global transformations towards a model* where TRAMIS automatically applies all transformations on all objects non consistent with a model, so that it becomes consistent (TRAMIS has entire knowledge).

a) Transformation of a relationship type into entity type

A relationship type R is replaced by an entity type of name R and by a binary relationship type R_i for each role r_i of the original relationship type. Cardinalities of entity type R in each R_i are 1-1, while cardinalities of the other roles are those of the original roles r_i. Identifier groups, textual description, statistics, technical notes and origins are derived from the original structure.

At the conceptual level, such a transformation is useful for 'promoting' a relationship type.

b) Transformation of an attribute into entity type

An attribute A (of cardinalities i-j) of an entity type E is put in a new entity type of name XE attached to E by a binary relationship type; the cardinality is deduced from the cardinality of A. The identifiers, textual descriptions, technical notes statistics and origins are computed from the original structure.

That transformation is useful during the design of the conceptual schema when one sees that an attribute must have attributes or need to be linked to different entity types.

c) Disaggregation of a compound attribute

A compound (non repetitive) attribute is replaced by its components, while respecting naming conventions.

This transformation may be useful to promote components of a compound attribute, or to restructure a compound attribute.

d) Aggregation of the attributes of a group

A collection of attributes is grouped to form a new compound attribute.

This tranformation allows one to restructure the attributes of an entity type or relationship type, or to modify the position of attributes.

D. Model compliance checking

When he feels it is necessary (according to the methodology he follows), the designer may ask for model compliance checking of specifications.

A general design workbench has to take into account the different models used in the steps of methodologies and variants for these models. It has therefore to offer some kind of parametrization of the design process according to the models used [Conc 90a].

As TRAMIS offers a single model, which 'covers' most of the models offered today, others models can be *defined by restricting* TRAMIS model. We saw that a model has a set of constructs and a set of rules specifying how constructs may be assembled. A new model is then defined by a set of constraints which limit the constructs available and their combination. For example, a standard relational model may be defined by specifying the following constraints:

- each entity type has at least one attribute;
- there cannot exist any relationship type;
- an attribute cannot be repetitive;
- an attribute cannot be compound.

TRAMIS offers a set of standard models. Others can be defined by the user with a language for the expression of constraints (see appendix 3 of [Conc 90a]); these constraints are stored in a text file accessible from TRAMIS.

TRAMIS offers functions for the diagnostic of the specification against these constraints. Transformations are used to make a schema compliant.

E. Production of executable descriptions

A schema may be translated in the data definition language of a data management system (DMS) (either a DBMS, or a file manager). If the translation cannot include all the aspects of the schema, because of limitations of the DMS, TRAMIS generates descriptions or procedures for the validations of these integrity constraints.

F. Reporting

Different aspects of the specifications can be reproduced on reports. Currently, TRAMIS offers three types of documentations:

- a detailed report;
- a global report;
- a complementary report of generation.

The latter, for the implementation, corresponds to validations of integrity constraints after generation. The detailed report presents all the specifications in five chapters: the schema, the entity types, the relationship types, the spaces (not studied here) and index of attributes. The global report presents statistics on the global schema.

G. Import/export

A textual specification language, called *ISL* (*Information System Specification Language*) allows the description of the specifications in a text file. This guarantees upward compatibility of specifications along TRAMIS evolution, and an opening towards other systems.

The incremental loader allows the integration in a same repository of different subsets of specifications of a same schema. The selective unloader allows the generation of (a part of) the specifications of a schema in a file of ISL format.

5.3.2. Technical level explanations

We have already said that the tools are working on the specification database.

In the following, we shall concentrate only on the algorithms working directly on the specification database. This simplification has been introduced for the ease of explanation and understanding; moreover it is

more interesting as the user might ignore TRAMIS low level, technical issues, which are not really interesting for the understanding of the logics of the procedures. The reader must be aware that the implemented architecture is *efficient*, as the most frequently used objects of the specification database are maintained in memory during a TRAMIS session.

Moreover, an *access module*, in the sense of [Hain 86a], has been designed for the development of the tool [Hain 89c]: its data model is a subset of the GAM, its primitives are also those of ADL and the composition rules are reduced to the minimum, thanks to the explicit notion of reference to a record (for each access function, there is an access to the first element verifying a condition, and another to the next element).

We shall not present here any algorithm. Indeed, we prefer to present, in next chapter, the modified functions with regard to the newly introduced generalization/specialization construct.

5.4. TRAMIS user interface and monitoring

5.4.1. User level explanations

TRAMIS interface follows the *Window Icon Mouse Pull-down menu* (*WIMP*) paradigm, but has only active texts. The different constructs and the schema are presented through specialized *windows*: the windows ENTITY, RELATIONSHIP, ATTRIBUTE, GROUP, COMPONENT (of a group), etc. Each window has a *menu bar* corresponding to operations which may be applied on constructs of that type. Each aspect of the constructs are described in a *subwindow* of the principal window.

The monitoring (figure 5.9) works as follows: according to the object-oriented principle of TRAMIS user interface, the user chooses an object (which then becomes the current object), and then selects - in a menu - a function to be performed on that current object. TRAMIS functions are grouped into classes [Conc 90a]:

- the global functions are working on the database schema;
- the functions of detailed elaboration are performed on constructs of that schema (entity type, relationship type, ...).

Appendix C contains a diagram explaining how monitoring is going on, (only the relevent menus are presented). We use the standard terminology (summarized in [Prov 89]) for *interactive objects*.

5.4.2. Technical level explanations

All human interface interaction is let to Windows. The user level explanation will be sufficient in this text.

Chapter 6

A generalization/specialization construct in TRAMIS: the category

In this chapter, we explain our choices concerning the definition of a generalization/specialization construct into TRAMIS: the category[1].

The idea of the generalization mechanism is to define a class of objects containing the objects of another class; the first class describes the objects in a less precise way than the latter does, i.e. it avoids some details. On the other hand, the specialization mechanism implies the definition of classes which are 'more concrete' than a given class; the objects of the first classes are included in the latter. The generalization/specialization mechanism allows then the introduction of abstraction levels, in order to master the numerous characteristics of an object of the UoD: we can describe it under one or another aspect, according to our viewpoint.

We shall now describe how this construct fits into TRAMIS architecture, model, processors, and user interface.

6.1. The category and TRAMIS architecture

We tried to define a construct which is:

- *minimal*, i.e. the category contains only the elements which have to be there, so that we can speak of a generalization/specialization construct;
- *consistent*, i.e. providing a unified approach to generalization/specialization;
- *useful*, i.e. powerful enough for a specialist but also easy to understand and to use by non-specialist, and using structures as 'natural' as possible.

Our choices have also been guided by the following 'constraint': the *transparency* of the integration into TRAMIS, i.e. someone who is not

[1] The reader is pleased not to confuse with Elmasari's category [Elma 85].

interested in the *is-a* extensions should be able to ignore them when working with TRAMIS.

A restricted model is not necessary a handicap. Indeed the simplicity to learn and use are important too. Moreover a restricted number of constructs reduces the problem of the representation choice. And if a model becomes too complex, the consistency of specifications is more difficult to define [Hain 89d].

Let us add that, as we are 'implementing' the model in a workbench, we tried to reach a compromise between usefulness, and development and user learning effort 'price to pay'. Our compromise runs as follows. We introduced only what could be transformed with as less new constraints as possible. A survey of the possible transformations to basic E-R constructs showed that we need at least the concept of global cardinality (see below), in default of the multidomain role. Therefore we restricted the introduced constructs (and functions) to those which can be replaced by TRAMIS constructs and/or the global cardinality constraint, but no more.

6.2. The category in TRAMIS model

Let us now define this construct as a component of TRAMIS model.

6.2.1. User level explanations

A. The category

a) Definition

We define the *category* of an entity type EG as a group of (direct) specific entity types ES_1, ES_2, ..., ES_n, $n \geq 1$, of that entity type (referred as the generic entity type). Remind that ES is a *specific entity type* of EG, or EG is a *generic entity type* of ES, or more simply, ES *is-a* EG if pop(ES) $\subseteq$ pop(EG) and ES is *more concrete* than EG (where pop(E) is the population of entity type E).

A category may also be characterized by *class inclusion constraints*:

- a *disjunction constraint* specifies that the specific entity types are disjoint, i.e. for each i and j $\in$ {1, 2, ..., n}, i <> j: $pop(ES_i) \cap pop(ES_j) = \{\}$;

- a *covering constraint* specifies that the specific entity types cover their generic entity type, i.e.

$$pop(EG) = \bigcup_{i=1}^{n} pop(ESi).$$

If both disjunction and covering constraints are specified, we say that the specific entity types form a *partition* of the generic entity type. If there is only a disjunction constraint, they form a *disjunction*, and if only a covering constraint is specified, they form a *cover*.

An entity type is allowed to possess (i.e. to be the generic entity type of) only one category - *single criterion specialization* - of any number of specific entity types - *multiple specialization* - and to belong to (i.e. to be specific entity type of) only one category - *single generalization* -. Of course, specific and generic entity types are belonging to the same schema. As explained in chapter 3, the *is-a* relation is transitive. Therefore, specific entity types of an entity type belonging to the category of another entity type, are also specific entity types of the latter: we speak in this case of *indirect* specific entity types (and *indirect* generic entity type).

b) Properties

We have also seen that the *is-a* relation is irreflexive and antisymmetrical; this implies that an entity type cannot belong to its category, nor to a category of one of its (direct or indirect) specific entity types. In consequence, the *is-a* relations implied by categories in TRAMIS model form an acyclic-directed graph; and as only single generalization is allowed, we deal with trees.

c) Notations

In this text, we shall use the following notation to represent a category of the entity type EG which is composed of specific entity types ES_1, ES_2, ..., ES_n, $n \geq 1$:

$$CAT\ (ES_1, ES_2, \ldots, ES_n \rightarrow EG).$$

The graphical representation is presented in figure 6.1 (*cic* indicates the class inclusion constraints if any, i.e. *D* for disjunction, *C* for cover, *P* for partition).

d) Remark

We do not provide a descriptive text for the categories, in order not to confuse the user about where semantics is to be put: we assume that semantic expressions about the generalization/specialization 'process' represented by the category are recorded either in the description of the generic entity type, or in the description of specific entity types. TRAMIS will not support any specialization assignement criterion, i.e. the condition on the entities of the generic entity type for their belonging to a specific entity type. Note however, that the user can record this criterion, in an informal way, in the description of each specific entity type, for instance.

e) Discussion

Could this generalization/specialization construct have been simpler? Allowing only one specific entity type per generic entity type would have been too restrictive as it is quite common to specialize an entity type in several specific entity types. If class constraints were not introduced, we would lose too much semantics, as explained in [Davi 89] for instance. Could it have been more complex? Of course, but we argue that with this construct which sounds clear and simple, we are able to model lots of cases. Let us now overview how more complex structures may be represented.

We restricted to one category per generic entity type. As a consequence, we cannot directly represent different specializations (under different criteria). But we may do it by an indirect way. For example, if we want to specialize books on their content basis, but also on their sales basis, we can do as presented in figure 6.2, i.e. to create intermediate entity types.

We compel all the specific entity types to be disjoint/covering or none of them. Doing so, we need not to care about the complex notions of maximal covering and minimal disjonction (see chapter 3). Using the same artifice as in the previous example, it is possible to precise class inclusion constraints between only certain specific entity types, while being sure than there is no problem of consistency.

We prohibit multiple generalization for the simplicity of the management of the workbench; also because multiple generalization is not always obvious for the user. However several authors argue that it becomes useful

in different applications. It is therefore a possible extension (if asked by the users). Note that in many cases, multiple generalization is introduced more for inheritance conveniences than for class constraints modelling purposes. In this case, we can promote the common specific entity type in the same generalization level as its generic entity types, and materialize property inheritance. Multiple generalization is suppressed, we keep the class inclusion aspects (but we loose conciseness in the schema).

The specialization criterion can only be represented in an informal way. This is another possible extension, however this 'constraint' seems to be a special case of redundant information, expressible integrity constraints.

As explained in chapter 3, we do not consider *may-be-a* relations as 'real' generalization/specialization constructs. Therefore, they are not introduced.

B. Inheritance

a) Definition

Inheritance is the way by which entity types can share their descriptive aspects (i.e. attributes, roles, relationships, and identifiant groups). We describe two basic inheritance 'operators': *upward inheritance* and *downward inheritance* (cf. chapter 3). They are applied on a couple of generic and specific entity types and they can concern:

- an attribute with its characteristics;
- a role with its characteristics;
- a group with its components.

Inherited characteristics are called *inherited*; the others are called *proper* if misunderstanding may arise. Keep in mind that inherited characteritics of an entity type may be inherited or proper characteristics of its direct generic entity type. These two operators may be combined to allow more powerful inheritance inferences, for example sideway inheritance.

b) Remark

In the next paragraph, we shall see that downward inheritance is supported for consultation purposes, and both downward and upward

inheritance are supported in transformations (these transformations will 'materialize' inheritance).

c) Discussion

We did not introduce redefinition nor inhibition constraints, because they would require the introduction of many other constraints in TRAMIS. However a textual description of them is possible: for instance, one might add them in the description of an attribute.

C. Categories and naming conventions

We keep the existing naming conventions; however this may lead to the fact that a proper attribute and an inherited attribute may have a same name: if this situation arises, the user must be aware of it (but there is no confusion for TRAMIS processors). Keeping these conventions will be useful in case of any integration of inhibition and redefinition constraints.

D. Categories and statistical model

By definition of the *is-a* relation, populations of entities have intersections. As a consequence, the model of static statistics, i.e. the average size of the population of entity types, has to verify certain constraints. The static aspects are explained here, while dynamic aspects will be explained with TRAMIS processors, as they are deeply tight.

One thing the reader has to be conscious of is that the population average sizes of entity types are not necessarily specified for each entity type of the project database; therefore, the following equations[2] have to deal with unspecified population:

(EQ_1) for each category CAT(ES_1, ES_2, ..., ES_n -> EG), $n \geq 1$,
$N_{ESi} \leq N_{EG}$, $1 \leq i \leq n$;

(EQ_2) for each category CAT(ES_1, ES_2, ..., ES_n -> EG), $n \geq 2$,
forming a disjunction,
$\sum_{i=1}^{n} N_{ESi} \leq N_{EG}$;

(EQ_3) for each category CAT(ES_1, ES_2, ..., ES_n -> EG), $n \geq 1$,

[2] In the following, we shall refer to them as the *statistical equations*.

forming a cover,

$$\sum_{i=1}^{n} N_{ESi} \geq N_{EG}.$$

As a consequence of (EQ_2) and (EQ_3), for each category $CAT(ES_1, ES_2, \ldots, ES_n \rightarrow EG)$, $n \geq 2$, forming a partition:

$$(EQ_4) \quad \sum_{i=1}^{n} N_{ESi} = N_{EG}.$$

In the latter case, two inference[3] rules can be applied:

(IR_1) if $N_{ES1}, N_{ES2}, \ldots N_{ESn}$ are specified, then N_{EG} is obtained by $\sum_{j=1}^{n} N_{ESj}$;

(IR_2) if $N_{ES1}, N_{ES2}, \ldots, N_{ESj-1}, N_{ESj+1}, \ldots, N_{ESn}$ and N_{EG} are specified then N_{ESi} is obtained by

$$N_{EG} - \sum_{\substack{i=1 \\ j \neq i}}^{n} N_{ESi}.$$

We have to be aware that some N_E may no be specified. Therefore, we assume that during the evaluation of equations, a missing N_E is replaced by the sum of the N_E of its direct specific entity types (and so forth recursively); and when the result of these 'assignments' always gives an unspecified value then 'unspecified' is assumed being equivalent to zero in equations (EQ_1) and (EQ_2), to ∞ for equation (EQ_3), and equation (EQ_4) is replaced by equations (EQ_2) and (EQ_3).

E. Category and identifier group

Due to the inheritance mechanism, an identifier group of an entity type is composed of the proper and the inherited attributes/roles.

F. A new integrity constraint: the global cardinality

In the next paragraph, we shall need a new integrity constraint in order to be able to preserve semantics of generalization/specialization when it is represented by basic concepts. We propose to define it here, as it will be part of the model.

[3] In the following, we shall refer to them as the statistical *inference rules*.

The global *cardinality*[4] i-j, $0 \leq i$, $1 \leq j$, $i \leq j$, for a group of roles and/or attributes of an entity type E specifies that each entity of E has to be associated with at least i values for these attributes and/or roles, and at the most j.

We may note that if a global connectivity i-j is defined on the group of n attributes/roles, $n \geq 2$, with cardinalities i_1-j_1, ..., i_n-j_n, then we must have that:

- $j \geq \sum_{k=1}^{n} j_k$;
- $i \geq \min_{1 \leq k \leq n} i_k$;
- for each i_k, $1 \leq k \leq n$, $j \geq i_k$.

6.2.2. Technical level explanations

How can we represent the category construct in TRAMIS specification database? As already stated, the specification database about which we proposed a particular interpretation in chapter 5 has been designed with two objectives: generality and precision of the concepts. Actually, a construct of generalization/specialization has already a representation in the specification database schema. But the proposition of [Hain 86b] specifies that a more detailed study is necessary and that it will probably lead to modifications of the proposed subschema.

The interpretation of the subschema of figure 6.3 is as follows. If an ENTITY_T represents a generic entity type, its specific entity types are classified in groups of subtypes (G_OF_SUBTYPE) via GST, defined by a criterion (CRITERION). The specific entity types corresponding to a same criterion are attached to the G_OF_SUBTYPE by relationship type SUBTYPE. A G_OF_SUBTYPE is also characterized by a name (GST_NAME), the indication that the population of the generic entity type contains, is strictly equal or is contained in the union of specific entity types populations (COVERING), the indication that specific entity types are disjoint or not (EXCLUSIVE), and parameters of free interpretation (PARAM1, PARAM2). Relationship type SUBTYPE is characterized by the indication that the specific entity type is strictly included or not in the generic entity type

[4] It will be another group, according to TRAMIS 'vocabulary'.

(TOTAL), and by parameters of free interpretation (PARAM1, PARAM2). Moreover, it is assumed that:

- an ENTITY_T cannot be (directly or indirectly) a SUBTYPE of itself;
- all ENTITY_Ts attached via GST and via SUBTYPE to a G_OF_SUBTYPE belong to the same SCHEMA.

The category constructs fits quite well in this schema, although we shall restrict it, because:

- as we consider only *is-a* relations, TOTAL is always equal to *true*;
- as only one generic entity type is allowed per entity type, cardinality of ENTITY_T in the SUBTYPE relationship type is 0-1;
- 'free' parameters are not necessary;
- the attribute CRITERION is not used;
- as only one category is allowed per entity type, GST_NAME brings no information;
- moreover the cardinality of ENTITY_T in GST is 0-1.

We obtain then the following subschema (figure 6.4). COVERING indicates if the category contains a covering constraint or not, and EXCLUSIVE states if the specific entity types are disjoint or not. The two constraints specified here above are always standing.

Moreover, we have to 'modify' the representation of identifier groups. Indeed, thanks to the inheritance mechanism, an identifier group may contain (downward) inherited roles/attributes. The less expensive solution consists in allowing a group to concern direct and inherited attributes/roles. This implies a modification of a constraint which was verified by a procedure (see previous chapter). Moreover, with this solution, it is quite easy to introduce extensions such as inhibition and redefinition constraints.

The global cardinality constraint will be represented by an ID_KEY_ORD (see chapter 5), where the TYPE is *Glob-card*. Parameters PARAM1 and PARAM2 are used to record respectively the minimum global cardinality and the maximum global cardinality. The latters have to verify the 'equations' relating them to 'local' cardinalities (see 6.2.1).

The corresponding GAM subschema is easily derived (see figure 6.5). Note however that we keep SUBTYPE as a record type, because we want to use the already defined access module.

6.3. The category and TRAMIS processors

In this paragraph we explain how the processors are modified or receive new functionalities in order to deal with the category.

6.3.1. User level explanations

A. Consultation

As explained in the previous paragraph, we shall apply automatically downward inheritance for consultation purposes. More precisely, an ON/OFF button is introduced. If this button is ON, then when consulting the attributes, the groups, and the relationship types concerning an entity type, we see not only the proper characteristics, but also the inherited characteristics from all generic entity types, i.e. the characteristics (proper and inherited) from the generic entity type are inherited by the specific entity type.

We can also consult the category of each entity type, and 'its' *is-a* hierarchy (i.e. its generic entity types). For the category, we can consult the class inclusion constraints.

B. Modifications

We must add new functions and modify existing ones.

a) Insertion of an *is-a* relation

This function consists in adding an *is-a* relation between two existing entity types, i.e. ES *is-a* EG, and if the generic entity type has no other specific entity type, its category is first created without any class inclusion constraint. In case of partition, statistics inference rules are applied.

It is however assumed that:

- ES does not belong to any category;
- EG is not a (direct or indirect) specific entity type of ES;
- statistical equations are verified.

b) Suppression of an *is-a* relation

This function consists in the removal of an *is-a* relation existing between two entity types, and consequently the deletion of the category if the generic entity type has then no more specific entity type. Since a group may contain inherited attributes/roles, they are suppressed from these groups.

c) Modification of the class inclusion constraints of a category

This function modifies class inclusion constraints of a category. Statistical equations must be verified however. If the category forms then a partition, statistical inference rules are applied.

d) Update of the population of an entity type

The new population has to verify statistical equations; moreover, statistical inference rules may be applied.

e) Deletion of an entity type

In current version of TRAMIS, an entity type can be suppressed only if it plays no (proper) role in a relationship type.

If that entity type is in an *is-a* hierarchy, its specific entity types become direct specific entity types of its generic entity type. If attributes of that entity type are belonging to groups of some specific entity types, they are suppressed from these groups (note that this cannot happen for roles).

f) Deletion of an attribute

If that attribute belongs to a group, it must be deleted from the group in which it belongs (eventually from a group of a specific entity type).

g) Deletion of a role

If that role belongs to a group, it must be deleted from the group in which it belongs (eventually from a group of a specific entity type).

h) Adding components to a group

A group may contain not only proper attributes/roles but also attributes/roles which are inherited from the generic entity type (if any).

C. Category and model compliance checking

TRAMIS has a single model which is a superset of the models the user and designer may encounter in practise. Therefore, it is useful to be able to define other models by enforcing restrictions.

Here are the set of constraints restricting the concept of category:

(C_1) <min>; <max>;
An entity type possesses from <min> to <max> categories. Currently, <min> = 0 and <max> = 1.

(C_2) <min>; <max>;
An entity type belongs to at least <min> categories and at most <max> categories per generic entity type. Currently, <min> = 0 and <max> = 1.

(C_3) <min>; <max>;
An entity type belongs to at least <min> categories and at most <max> categories of different entity types. Currently, <min> = 0 and <max> = 1.

(C_4) <min>; <max>;
A category contains from <min> to <max> specific entity types. Currently, <min> = 1 and <max> = *N*.

(C_5) Each category must include the disjunction constraint.

(C_6) Each category must include the covering constraint.

D. Category and import/export

The import and export processes must take into account the concept of category, which has already been defined in ISL syntax (see appendix 2 of [Conc 90a]); a modified version is as follows:

- ET-def::= entity-type ET-name ET-shortname;
  [origin object-name] [date date]
  [is-in CATEGORY-name]$^{0-1}$
  CATEGORY-def$^{0-1}$ description^{0-*}
  GROUP-def^{0-*}

- CATEGORY-def::= categories [covering] [exclusive]
  specific spec-ET-def[1]...*
  end categories
- spec-ET-def::= ET-name.

The definition of a group is modified as follows:

- GROUP-def::= group GROUP-CODE;
  [id] [primary] [key]
  [glob-card Min Max]
  RELATION-block[0]...*
  [GR-comp-block]
  STAT-block[0]...*
  ACTION-block[0]...*
  end-group
- Min:= integer from 0 to 99999
- Max:= integer from 1 to 99999.

The loader and the unloader have to take these specifications into account.

E. Category and executable schema generation

Since all generation rules for DBMSs data definition languages are defined on the basic E-R model, and since *is-a* relations cannot be translated 'directly' in most current DBMSs, we assume that before code generation, all *is-a* relations have already been translated into basic E-R concepts. Therefore that processor need not to be modified, except that procedures are generated for global cardinality groups.

F. Category and report generation

The reporting function about an entity type shows now the following new elements:

- the category of that entity type (with the class inclusion constraints and the name of the specific entity types);
- the name of its generic entity type;
- if the automatic inheritance button is ON, the inherited attributes, roles and identifier groups are presented too.

Moreover, the attribute index shows now all pages in which the attributes appear:

- where they are defined;
- where they are inherited (if the inheritance button is ON).

The global cardinality constraints are shown as the other groups.

G. Transformations of a category

Transformations may be useful for different purposes:

- in the database design mapping process to eliminate categories from a schema, and obtain a basic E-R schema;
- in the schema drawing process.

The functions explained here may be of interest for these 'processes'.

We shall develop transformations for the elimination of categories from a schema. There are mainly three ways to suppress a category of a generic entity type, as already mentioned in chapter 4. We can apply the upward inheritance mechanism, and represent the *is-a* semantics by keeping only the generic entity type; conversely, we can apply downward inheritance and keep only the specific entity types. Finally, we can simply represent the *is-a* relations of a category by relationship types. After the presentation of elementary transformations, we present also the global transformations[5]. The other transformations are typically useful for an 'a posteriori' introduction of categories, and for schema reorganization.

Before presenting these new transformations, we must precise that:

- we assume that they are not applied on schemata containing 'non conceptual' constructs;
- the transformations explained in chapter 5 work properly even if there are categories in the schema;
- the latter transformations must consider the global cardinality groups too.

All transformations must, by definition, provide both syntactical correctness (i.e. the schema obtained after transformation of a schema compliant the model is still compliant to the model, here the E-R model)

[5] Transformation towards models are not presented here.

and semantic equivalence. In the following, syntactical correctness will be verified (although, naming conventions could be mistreated; in this case, we assume that the user is asked to enter a 'good' name). Semantic equivalence verified and; statistical aspects are also preserved[6].

a) Transformation consisting in grouping entity types in a category of a new entity type

This transformation consists in the creation of an entity type considered as the generic entity type of a group of entity types, say ES_1, ES_2, ..., ES_n, $n \geq 1$, (which do not belong to any category). Moreover, upward attribute/role inheritance is applied for the common attributes/roles proposed by the user. Attributes are *common* if they have the same type, the same length, the same number of decimals, the same number of components and if their components are common. Roles are *common* if they have the same cardinality, if their relationship type have the same degree, if the other roles are defined on the same entity types and their cardinality is 0-N (or a global cardinality is defined on them). The category of the newly created entity type forms a partition composed of ES_1, ES_2, ..., ES_n. These upward inherited attributes/roles include in their technical note and description what was contained in the technical note and description of the correspondant attributes/roles. In the technical note of EG, it is noticed that it results from that transformation. Statitistic inference rules are applied for that new category.

Its inverse transformation consists in representing a category by the sole specific entity types (see below). This transformation is useful when several existing entity types must be generalized (see chapter 4).

b) Representation of a category by the specific entity types

This transformation applies on a category CAT(ES_1, ES_2, ..., ES_n -> EG), $n \geq 1$, forming a partition[7]. The category and the generic entity type EG are removed. Each attribute of EG is inherited by each specific entity

[6] This justifies that the preconditions of certain transformations are 'heavy', as far as statistical aspects are concerned. If one prefers a less vigorous approach, he may 'drop' the aspects concerning statistics in the preconditions (and record statistical information in the technical notes).

[7] If the specific entity types did not cover the generic entity type, we would need the creation of an 'artificial' specific entity types (containing entities which do not belong to specific entity types). If the specific entity types were not disjoint, we would need additional constraints to represent possible overlapping specific entity types, as explained in paragraph 4.2.

type. The roles played by EG are replaced by a multidomain roles[8] played by ES_1, ES_2, ..., and ES_n. The generic entity type cannot have any identifier group[9]; the global cardinality groups are inherited by each specific entity type. The technical note (and origin) of inherited attributes and relationship types concerned by the inherited roles contain an annotation of that transformation. Each specific entity type 'receives' the description of EG, and its technical note. If EG belonged to a category, it is replaced by its specific entity types ES_1, ES_2, ..., ES_n in this category. This transformation is not allowed if the average size of EG is specified, while the average population sizes of the specific entity types[10] are not specified.

The inverse of this transformation is the previous one. It is essentially used for the removal of *is-a* constructs from a schema. Note that this representation of generalization/specialization constructs should be avoided when relationship types are defined on EG, in order not to have to deal with global cardinalities.

c) Transformation of a category to include the covering constraint

This transformation applies on a category CAT(ES_1, ES_2, ..., ES_n -> EG), $n \geq 1$, which does not include the covering constraint. It consists in the creation of a new entity type OTHER_EG, specified as belonging to that category, and in the modification of the category to include the covering constraint. The name, the shortname, the description, the technical note, and the origin of this new entity type are derived from characteristics of EG. If the modified category forms a partition, then statistic inference rules are applied, i.e. the population average size N_{OTHER_EG} of OTHER_EG

---

[8] Indeed, since the construct of multidomain role is not in TRAMIS basic model, the replacement by a multidomain role is actually a replacement by the replacement of a multidomain role, i.e. by multiplication of relationship types [Coll 88].
Given a relationship type R of degree n, $n \geq 2$, where:
- the role r is multidomain role played by E_1, E_2, ..., E_m ($m \geq 2$);
- the other concerned entity types are EE_1, EE_2, ..., EE_{n-1}.

To eliminate that multidomain role, we replace R by m relationship types R_1, R_2, ..., R_m. Each R_i ($1 \leq i \leq m$) is a 'copy' of R, except that:
- the multidomain role r is replaced by a role r_i played by E_i (its cardinality is that of r);
- the roles of each EE_j ($1 \leq j \leq n-1$) in R_1, R_2, ...R_n are concerned by a global cardinality, the values of which are those of the cardinality of EE_j in R.

[9] Otherwise, we need the introduction of a new integrity constraint, i.e. the *global identifier constraint*. Let I_i be an identifier group of entity type E_i ($1 \leq i \leq n$). If a global identifier constraint is specified on (I_1, I_2, ..., I_n) for (E_1, E_2, ..., E_n) then given a value for I_i ($1 \leq i \leq n$) there is at the most one entity of $E_1 \cup E_2 \cup ... E_n$ with that value.

[10] So we are sure to keep the statistical aspects.

is given by $N_{OTHER\_EG} = N_{EG} - \sum_{i=1}^{n} N_{ES_i}$ if the latters are specified, otherwise it is unknown.

The reverse transformation consists simply in the deletion of OTHER_EG and in the removal of the covering constraint. This transformation may be useful in order to meet the precondition of other transformation (for instance, the previous one), or to be compliant with a restricted model which asks for categories with the covering constraint.

d) Transformation of a category to include the disjunction constraint

This transformation applies on a category CAT(ES_1, ES_2, ..., ES_n -> EG), $n \geq 2$, which does not include the disjunction constraint (and for which the average sizes of population are not specified)[11]. The specific entity type cannot have identifier groups[12]. It consists in the creation of $\sum_{i=1}^{n} C_i^n$ new entity types, specified as specific entity types of EG, and in the modification of the category so that it includes the disjunction constraint. The C_2^n first entity types represent $ES_i \cap ES_j$, i and j $\in \{1, 2, ..., n\}$, $i <> j$ (this is specified in the technical note). The attributes, roles[13] and groups of these entity types are those of ES_i added to those of ES_j. The C_3^n next entity types represent $ES_i \cap ES_j \cap E_k$, i, j and k $\in \{1, 2, ..., n\}$, $i <> j$, $i <> k$, $j <> k$ (this is specified in the technical note). The attributes, roles and groups of these entity types are those of ES_i added to those of ES_j and ES_k... The name and shortname are derived from the original structure. Their origin is EG, i.e. the entity type to which the category belongs. The average population sizes are let unspecified.

The inverse transformation is to delete the newly created entity types and to remove the disjunction constraint. This transformation may be useful in order to meet the precondition of other transformation, or to be

[11] So we can preserve statistical equivalence.

[12] So we do not need the introduction of global identifiers.

[13] This means that roles are replaced by multidomain roles, i.e. a role of ES_i, $1 \leq i \leq n$, is replaced by a multidomain role played by ES_i and the new entity types representing $ES_i \cap ES_j$, $ES_i \cap ES_j \cap ES_k$, ..., and $ES_1 \cap ES_2 \cap ... \cap ES_n$.

compliant with a restricted model asking for categories with the disjunction constraint.

e) Transformation consisting in the creation of a partition of an entity type according to a set of optional roles/attributes

Given an entity type E without any category, and given a set of optional roles/attributes (belonging to no group), two entity types E_1 and E_2 are created which belong to a newly created category forming a partition of E. E_1 contains the attribute(s)/role(s) of that set (which become mandatory), while E_2 contains entities of E with no value for the attribute(s)/role(s) of that set. This is specified in their technical note. Their origin is E. Their average population size is let unspecified.

The inverse transformation consists in the representation by the generic entity type (see below). It is useful when we see that a set of optional attributes/roles of an entity type 'hides' the existence of specific entities (see chapter 4), i.e. a 'null' value for these attributes/roles means 'does not exist' (and not 'unknown').

f) Representation of a category by the generic entity type

This function applies on a category CAT(ES_1, ES_2, ..., ES_n -> EG), $n \geq 1$, the specific entity types of which possess at the most one mandatory role or only attributes[14] and do not belong to any category[15] nor possess any identifier group[16]. The category and the specific entity types are deleted. The possible role of a specific entity type is put in the generic entity type, becomes optional; its name is changed to the name of the specific entity type, the technical note of its relationship type contains an annotation of that transformation: entities of the generic entity type playing this role are those of the corresponding specific entity type. The possible attributes of a specific entity type are grouped in the generic entity type into a new optional and simple attribute which has the name of the specific entity type it represents (this is noted in its technical note), i.e. entities having a value for that compound attribute are those

[14] This restriction has been introduced in order to avoid the introduction of too many integrity constraints. The attribute or the role can represent by itself the specific aspect of the generic entities.

[15] Because only one category is allowed per entity type, categories cannot be 'inherited'.

[16] So we need not to deal with partial identifier groups, i.e. with an identifier for only a subset of the entities of the entity type.

of the specific entity type it represents. If specific entity types do not have any attribute nor role, they are represented in the generic entity type by a boolean attribute with their name (this is noted in its technical note). In the last two cases, the frequency of the new attribute is given by N_{ESi} / N_{EG}, where ES_i, $1 \leq i \leq n$, is the concerned specific entity type (therefore, if N_{ESi} is specified, N_{EG} must also be specified in order to keep statistical information). The technical note and the description of the specific entity types are added to the technical note and description, respectively, of the generic entity type; the latter contains also an annotation of the transformation. If there is only one specific entity type covering the generic entity type, then the role or attribute representing it becomes mandatory. If more than two specific entity types form a disjunction, then we specify a global cardinality 0-1 concerning the roles/attributes which represent them. If they form a cover, then we specify a global cardinality 1-n concerning the roles/attributes which represent them. And if they form a partition, then we specify a global cardinality 1-1 concerning the roles/attributes representing them.

This transformation is another way to suppress *is-a* relations from a schema. It is essentially intended for situations when specific entity types have only a few proper characteristics.

g) Transformation consisting in the creation of a specific entity type

Given an entity type E which does not have a category with class constraints and given one or several optional attribute(s)/role(s), a new specific entity type is created (the category is created if it is its first specific entity type) which contains that (these) optional attribute(s)/role(s). If they belong to a group, the latter is inherited by the newly created entity type. The technical note of that entity type contains an annotation of this transformation, and its origin is EG.

This transformation is useful to represent multiple criterion specialization, as explained in 6.2.1. The inverse transformation is presented hereafter.

h) Transformation consisting in the suppression of a specific entity type

Considering an entity type EG for which a category exists but without any class constraint, given a specific entity type (for which the average population size is not specified) with optional attributes/roles, that specific entity type is deleted (and consequently the category, if it is the last specific entity type). Its attributes/roles are put in the generic entity type. Its groups are also inherited by the generic entity type.

This transformation is the inverse of the precedent one, and another alternative to the representation by the generic entity type.

i) Representation of a category by relationship types

This function applies a category CAT(ES_1, ES_2, ..., ES_n -> EG), $n \geq 1$. Each *is-a* link between EG and ES_i ($1 \leq i \leq n$) is replaced by a relationship type R_i between EG and ES_i. ES_i plays a role of cardinality 1-1. If there is only one specific entity type with the covering constraint, then the cardinality of the role played by EG is 1-1 too; otherwise, its cardinality is 0-1. If there are at least two specific entity types:

- if they form a disjunction, then a global cardinality 0-1 is specified between these new roles played by EG;
- if they form a cover, then a global cardinality 1-n is specified between these new roles played by EG;
- if they form a partition, then a global cardinality 1-1 is specified between these new roles played by EG.

The origin of each new relationship R_i type is EG (meaning that they come from the elimination of its category). Their technical note contains the fact they represent an *is-a* relation. Statistics of the relationship types are obtained from population of the concerned entity types, as explained in chapter 5. This transformation is the last alternative for the elimination of *is-a* relations from a schema. The inverse transformation consists in replacing the relationship types by the *is-a* relations of a category; the global cardinalities are replaced by the corresponding class constraints. It seems to be the most 'natural' one, however it implies that UoD objects are represented in different, disjoint entity types (see 4.2).

j) Global transformation: representation of the categories by the sole specific entity types

This transformation aims at progressively eliminating *is-a* relations from the schema by representing only the specific entity types. It proceeds in a top-down fashion i.e. starting from entity types which do not belong to a category (in order to keep trace of the hierarchy in the structure of the entity types attributes). Due to the precondition of the corresponding elementary transformation it may be possible that some categories cannot be transformed in this way.

k) Global transformation: representation of the categories by the sole generic entity types

This transformation progressively eliminates *is-a* relations from the schema by keeping only the generic entity types. It proceeds in a bottom-up way, i.e. starting from entity types which do not possess a category (in order to keep trace of the hierarchy in the structure of the entity types attributes). Due to the precondition of the corresponding elementary transformation it may be possible that some categories cannot be transformed in this way.

l) Global transformation: representation of the categories by relationship types

This transformation progressively eliminates the *is-a* relations from a schema: they are replaced by relationship types.

6.3.2. Technical level explanations

On this level, we are interested in the design of the main algorithms (presented in appendix E); they are written in ADL [Hain 86a] and are working on the GAM schema of the specification database where the category construct is represented. The choice of this language was obvious since we are working on a GAM schema. We however restrict ourselves to the primitives which are provided by the access module implemented in TRAMIS (cf. 5.3.2). These algorithms are consistent to the virtual DMS constituted by the access module. We tried, of course, to reduce the number of accesses to the database.

6.4. The category in TRAMIS user interface

6.4.1. User level explanations

How does the category fit in user interface of TRAMIS?

Our objective was to make a 'transparent' concept[17], so that only users interested in it will see it: it is a 'secondary' construct. Appendix D contains the modifications of the current user interface monitoring (explained in appendix C). We see that these modifications are minor, except the fact transformations are defined in the ENTITY menu. Note also that the user can specify the generic entity type of the current entity type, or specify the specific entity type(s) of the current entity type.

6.4.2. Technical level explanations

As explained in chapter 5, we assume that user level explanations are sufficient here.

[17] This choice is guided by current remarks on TRAMIS, which is considered as relatively complex... due to the richness of its concepts.

Conclusion

Main ideas

In this dissertation we studied generalization/specialization abstractions structures.

After the recall of what is an abstraction mechanism, we overviewed how these structures fit during the development of software, and compared their impact on three major areas in computer science, namely programming languages, artificial intelligence and databases. We particularly emphasized that generalization/specialization is used, along with its inheritance mechanism, for enhancing reusability of programs. Knowledge representation and knowledge manipulation are based on the generalization/specialization inferences, either strict inheritance, or default inheritance. Conceptual models but also implementation models (of object-oriented DBMSs) can profit from generalization/specialization in the perspective of capturing more semantics about the UoD.

Then, after a review of database design concepts, we thoroughly analysed, within a common framework (the GER model), how generalization/specialization structures have been introduced in semantic data models. The main elements are:

- the *is-a* relation between entity types (also proposed for relationship types and attributes) which records the abstraction mechanism used in the cognitive process;
- inheritance which allows for characteristics sharing amid types, and consequently enhances schema flexibility and conciseness;
- class constraints which specify how classes intersect;
- the criteria (and the possibility to include derived information);
- the *may-be-a* relation.

We have also overviewed how these elements have been combined in several models. We then confronted database design activities and generalization/specialization structures. That analysis aimed at providing us with a better idea of what can be used in any model.

From that study, we derived a subset and integrated it in TRAMIS, a database design workbench. That integration has been conducted at different levels (and for two kinds of readers, i.e. a user and an implementor):

- in TRAMIS architecture;
- in TRAMIS model;
- in TRAMIS processors;
- in TRAMIS user interface.

Other works

Analysis of generalization/specialization concepts have already been done (see, for example, [Coll 88], [Hain 89d] or [Spac 89]). We decided to provide such a study, stressing elements which were not analysed previously (the criteria, for example), and comparing with artificial intelligence and programming languages concepts. That study was also necessary for helping us to define the subset to be integrated in the workbench.

On the other hand, we did not find any paper on the problem of integrating a new construct in a database design workbench. However, our experience showed us that this task is for many reasons far from being obvious. If we need not think about the general framework, as it is the case when we start from scratch, it is however difficult in the sense that we have to be compliant with the existing.

Evaluation

The process of including a construct into a model (or the definition of a new model) is subject to criticism. Integrating it in a workbench adds further difficulties. The important thing is to propose a concept being really closed to mind structuring structures. That was our aim. We tried to provide a construct being consistent, minimal, useful (easy to understand and to use).

As explained in these pages, we defined the category construct as a consequence of both a theoretical study, which emphasized the different elements proposed in the literature, and also practical considerations, i.e. constraints related to TRAMIS (and its acceptance in the 'pratical world'). How did we manage it?

On the one hand, we started by overviewing the different concepts of generalization/specialization. We then studied TRAMIS. We saw that these

concepts were relatively easy to integrate in the TRAMIS model. Then, because most current DBMSs cannot deal with extensions like the generalization/specialization constructs, we analysed how extended constructs can be mapped to basic ones. This put in evidence a large number of integrity constraints we tried to classify (global identifier, partial identifier, global cardinality, ...), but also the difficulty of the algorithms (for multiple generalization, for example). An evaluation of the impact of the introduction of new integrity constraints in TRAMIS (by analyzing how they can be incorporated in the model, how they go through the existing transformations, and how code can be generated) helped us to decide several restrictions on both elements of the generalization/specialization construct (for example, we eliminated the redefinition/inhibition constraints of inheritance, and the specialization criterion) and on the transformations (for example, the representation by the generic entity type has been simplified). We finally arrived to the proposed category construct and the set of functions working on it (note that several existing functions needed to be modified too).

On the other hand, we undertook a small-scale field study, namely during the teaching of an one-week database design course to DEC engineers. We introduced them to the basic E-R approach but we also explained a generalization/specialization construct (which was a superset of the category). We saw, through exercises, that the proposed construct was knowledgeable. However, that field study aspect should be broadened, because:

- if we saw that the minimal subset was all right, we cannot argue that database designers will find it sufficient[1];
- the field study concerned a generalization/specialization construct 'outside' the 'world of TRAMIS'.

Future work

A field study concerning the evaluation of the generalization/specialization in TRAMIS will probably lead to an improvement of our proposals. Such a field study should 'define' different classes of users. It seems that an 'a posteriori' study is only worth, as far as the user interface problem is concerned, at least. That study could include an evaluation of TRAMIS itself, as well as an evaluation of E-R-based methodologies. Indeed, looking at an experience on teaching in the industry and university,

[1] If we had done that analysis, we do not however think that its integration within TRAMIS was possible for the reasons mentioned before (i.e. constraints of TRAMIS).

Nijssens [Nijs 86] argues that there is a need to exchange experience of teaching conceptual schema design as it gradually becomes part of the core of computer science. So few scientific (or at least systematic) studies have been carried out and reported concerning experiences of using methodologies on realistic, practical cases [Bube 87].

Model enhancement is a never-ending task. We proposed a minimal subset for generalization/specialization. Some designers will perhaps need extensions to the category. But in case of extensions, two questions should be addressed:

- why this extension?
- what is the original contribution of it?

Among the mechanisms which could extend the expressive power of a schema are certainly the integrity constraints and derived schema components aspects of generalization/specialization.

We restricted ourselves to static, structural aspects of generalization/specialization in our study arguing that only them are used in TRAMIS. We stressed their semantic power of expression. We did not analyse the 'application' aspect, nor the manipulation languages [Czed 90] [Spac 89] [Zani 89]. They should be analysed if manipulations aspects are considered in subsequent versions of TRAMIS.

We studied the use of generalization/specialization for database design, considering that DBMSs are not able to deal with them. Current researchs focus on object-oriented DBMSs, as explained in chapter 1. Such DBMSs are able to record certain generalization/specialization constructs. We think that current methodologies should be adapted then. Approaches using the DBMS model for the logical step will not need lots of modifications. On the other hand, approaches introducing an intermediate model, like the GAM, will have to extend it. In TRAMIS, this will imply the analysis of access and update actions on the category.

The development tasks proposed in chapter 2 were intended for a new information system. But in the future, adaptation of existing information systems will play a more proeminent role, as there will be more and more existing ones. The reorganization work will be more and more important. As different softwares are built in different ways, an approach to combine

them is to rebuild (or build?) the conceptual schema. And there, generalization/specialization may help too, as a *reverse engineering tool*.

During the analysis of the transformation of the generalization/specialization structures into basic E-R constructs, we were confronted with the need of integrity constraints (in order to keep the semantics of the 'extended' concepts of generalization/specialization). In our mind, a systematic study, trying to identify the most frequent ones and so provide a 'framework' for integrity constraints, will be of benefit for the expression power of the E-R model. This work is certainly worth writing a dissertation.

From these observations, we conclude that our contribution to the analysis of generalization/specialization structures in computer-aided database design could be extended in order to 'manage' the modelling tasks of computer science still better...

References

List of references

[Abri 74]
Abrial, J. R., "Data semantics" in Klimbie, J. W., Koffeman, K. L. (eds.), *Database management*, North-Holland, 1974, pp. 1-60

[Afsa 86]
Afsamanesh, H., Mc Leod, D., "A framework for semantic database models" in [Aria 86], pp. 149-168

[Alte 90]
Altenkrueger, D. E., "KBMS: aspects, theory and implementation", *Information Systems*, vol. 15, nr. 1, 1990, pp. 1-7

[Amer 87]
America, P., "Inheritance and subtyping in a parallel object-oriented language" in [Bezi 87], pp. 234-242

[Aria 86]
Ariav, G., Clifford, J. (eds.), *New directions for database systems*, Ablex Publish. Corp., Norwood, New Jersey, 1986, 269 p.

[Bane 87]
Banerjee, J., Chou, H.-T., Garza, J. F., Kim, W., Woelk, D., Ballou, N., Kim, H.-J., "Data model issues for object-oriented applications" in Stonebraker (ed.), *Readings on database systems*, pp. 445-456

[Bati 82]
Batini, C., Lenzerini, M., Santucci, G., "A computer-aided methodology for conceptual database design", *Information Systems*, vol. 7, nr. 3, 1982, pp. 265-280

[Bati 86]
Batini, C., Lenzerini, M., Navathe, S. B., "A comparative analysis of methodologies for database schema integration", *ACM Computing Surveys*, vol. 18, nr. 14, December 1986, pp. 323-363

[Bati 88a]
Batini, C. (ed.), *Entity-relationship approach: a bridge to the user*, Proceedings of the 7th International Conference on Entity-Relationship, North-Holland, Amsterdam, 525 p.

[Bati 88b]
Batini, C., Di Battista, G., "A methodology for conceptual documentation and maintenance", *Information Systems*, vol. 13, nr. 3, 1988, pp. 297-318

[Batr 90]
Batra, D., Hoffer, J. A., Bostrom, R. P., "Comparing representations with relational and EER models", *Communications of the ACM*, vol. 33, nr. 2, February 1990, pp. 126-139

[Berg 88a]
Bergamaschi, S., Bonfatti, F., Cavazza, L., Sartori, C., Tiberio, P., "Relational data base design for the intensional aspects of a knowledge base", *Information Systems*, vol. 13, nr. 3, 1988, pp. 245-256

[Berg 88b]
Bergamaschi, S., Cavedoni, L., Sartori, C., Tiberio, P., "On taxonomic reasoning in the E/R environment" in [Bati 88a], pp. 301-312

[Berm 86]
Berman, S., "A semantic data model as the basis for an automated database design tool", *Information Systems*, vol. 11, nr. 2, 1986, pp. 149-165

[Bezi 87]
Bezivin, J. (ed.), *ECOOP'87: European conference on object-oriented programming*, Springer, Berlin, 1987, 273 p.

[Blah 88]
Blaha, M. R., Premerlani, W. J., Rumbaugh, J. E., "Relational database design using an object-oriented methodology", *Communications of the ACM*, vol. 31, nr. 4, April 1988, pp. 414-427

[Boda 89]
Bodart, F., Pigneur, Y., *Conception assistée des systèmes d'information - Méthode, modèles, outils*, Masson, Paris, 1989, 302 p.

[Borg 84]
Borgida, A., Mylopoulos, J., Wong, H. K. T., "Generalization/specialization as a basis for software specification" in [Bro 84a], pp. 87-114

[Borg 85]
Borgida, A., "Features of languages for the development of information systems at the conceptual level", *IEEE Software*, vol. 2, nr. 1, January 1985, pp. 63-72

[Borg 88]
Borgida, A., "Modeling class hierarchies with contradictions" in Boral, H., Larson, P.-A. (eds.), *Proceedings of the SIGMOD international conference on management of data*, ACM, 1988, pp. 434-443

[Bouz 84]

Bouzeghoub, M., Gardarin, G., "The design of an expert system for database design" in Gardarin, G., Gelenbe, E. (eds.), *New applications of databases*, Academic Press, London, 1984, pp. 203-223

[Bouz 86]

Bouzeghoub, M., Metais, E., "SECSI: an expert system approach for database design" in [Kugl 86], pp. 251-257

[Brac 83]

Brachman, R. J., "What IS-A is and isn't: an analysis of taxonomic links in semantic networks", *IEEE Computer*, 1983

[Brac 85]

Brachman, R. J., Levesque, H. J., *Reading in knowledge representation*, Morgan Kaufmann Publ., Los Altos, California, 1985, 571 p.

[Brac 89]

Brachman, R. J., Schmolze, J. G., "An overview of the KL-ONE knowledge representation system" in [Mylo 89], pp. 207-229

[Bria 85]

Briand, H., Habrias, H., Hue, J.-F., Simon, Y., "Expert system for translating an E-R into databases" in [Chen 85] pp. 199-206

[Brod 80]

Brodie, M. L., "Data abstraction, databases and conceptual modelling", summary of Brodie, M. L., Zilles, S. N. (eds.), *Proceedings of the workshop on data abstraction, databases and conceptual modelling*, ACM, 1980, pp. 105-108

[Brod 81]

Brodie, M. L., "Association: A database abstraction for semantic modelling" in [Chen 81a], pp. 583-608

[Brod 84a]

Brodie, M. L., Mylopoulos, J., Schmidt, J. W. (eds.), *On conceptual modelling - Perspectives from artificial intelligence, databases, and programming languages*, Springer-Verlag, New York, 1984, 503 p.

[Brod 84b]

Brodie, M. L., "On the development of data models" in [Bro 84a], pp. 19-48

[Brod 86]

Brodie, M. L., Mylopoulos, J., "Knowledge bases and databases: semantic vs. computational theories of information" in [Aria 86], pp. 186-218

[Bube 87]

Bubenko, J. A., "Information analysis and conceptual modeling" in [Pard 87], pp. 141-192

[Carb 80]
Carbonnel, J. G., "Default reasoning and inheritance mechanisms on type hierarchies" in *Proceedings of the ACM SIGMOD conference on the management of data*, ACM, 1980, pp. 107-109

[Cata 88]
Catarci, T., Ferrara, F. M., "OPTIM_ER: an automated tool for supporting the logical design within a complete CASE environment" in [Bati 88a], pp. 231-246

[Ceri 81]
Ceri, S., Pelagatti, G., Bracchi, G., "Structured methodology for designing static and dynamic aspects of data base applications", *Information Systems*, vol. 6, nr. 1, 1981, pp. 31-45

[Chen 76]
Chen, P. P.-S., "The entity-relationship model - Toward a unified view of data", *ACM Transactions on Database Systems*, vol. 1, nr. 1, March 1976, pp. 9-36

[Chen 81a]
Chen, P. P.-S. (ed.), *Entity-relationship approach to information modeling and analysis*, ER Institute, 1981

[Chen 81b]
Chen, P. P.-S., "A preliminary framework for entity-relationship models" in [Chen 81a], pp. 19-28

[Chen 83]
Chen, P. P.-S., "ER - A historical perspective and futures directions" in [DaJa 83], pp. 71-77

[Chen 85]
Chen, P. (ed.), *Entity-relationship approach: the use of E-R concepts in knowledge representation*, North-Holland, 1985

[Cive 88]
Civelek, F. N., Dogac, A., Spaccapietra, S., "An expert system approach to view definition and integration" in [Bati 88a], pp. 97-117

[Codd 79]
Codd, E. F., "Extending the database relational model to capture more meaning", *ACM Transactions on Database Systems*, vol. 4, nr. 4, April 1979, pp. 397-434

[Coll 88]
Collart, N., Joris, M., *Etude théorique et pratique d'extensions au modèle entité-association*, Master Thesis, Institut d'Informatique, FUNDP, Namur, 1988

[Conc 90a]
Concis, FUNDP, *TRAMIS, un atelier de conception de bases de données - Manuel de référence version 1.0*, November 1990

[Conc 90b]
Concis, *Présentation technique de TRAMIS*, Argenteuil, January 1990

[Czed 90]
Czedo, B., Elmasri, R., Rusinkiewicz, M., Embley, D. W., "A graphical data manipulation language for an extended entity-relationship model", *IEEE Computer*, vol. 23, nr. 3, March 1990, pp. 26-36

[Dard 89]
Dardenne, A., Delcourt, B., Dubisy, F., van Lamsweerde, A., *The KAOS project: knowledge acquisition in automated specification of software*, Research Paper, Institut d'Informatique, FUNDP, Namur, 1989

[Date 86]
Date, C. J., *Relational database - Selected writings*, Addison-Wesley, 1986

[DaJa 83]
Davis, C. G., Jajodia, S., Ng, P. A., Yeh, R.T. (eds.), *Entity-relationship approach to software engineering*, ER Institute, Elsevier Science Publishers B. V., North-Holland, 1983

[Davi 89]
Davis, J. P., Bonnel, R. D., "Modeling semantics with concept abstraction in the EARL data model" in *Proceedings of the 8th International Conference on Entity-Relationship*, 1989, pp. 102-117

[Delo 82]
Delobel, C., Adiba, M., *Bases de données et systèmes relationnels*, Bordas, Paris, 1982, 449 p.

[DiBa 89]
Di Battista, G., Lenzerini, M., "A deductive method for entity-relationship modelling" in Alpes, P. M. G., Wiederhold, G. (eds.) *Proceedings of the 15th International Conference on Very Large Data Bases*, Kaufmann, Palo Alto, California, 1989, pp. 13-21

[Ditt 90]
Dittrich, K., "Object-oriented database systems: the next miles of the marathon", *Information Systems*, vol. 15, nr. 1, 1990, pp. 161-167

[Doga 81]
Dogac, A., Chen, P. P.-S., "Entity-relationship model in the ANSI/SPARC framework" in [Chen 81a], pp. 361-378

[Dols 88]
Dols, P., *Data bases conception and logical design*, Draft Course Support, Digital Equipment Corporation, Maynard, August 1988

[Duco 87]
Ducournau, R., Habib, M., "On some algorithms for multiple inheritance in object-oriented programming" in [Bezi 87], pp. 243-252

[Duco 89]
Ducournau, R., Habib, M., "La multiplicité de l'héritage dans les langages à objets", *Technique et Science Informatiques*, vol. 8, nr. 1, 1989, pp. 41-62

[Elma 85]
Elmasri, R., Weeldreyer, J., Hevner, A., "The category concept: an extension to the entity-relationship model", *Data & Knowledge Engineering*, vol. 1, nr. 1, 1985, pp. 75-116

[Equi 87]
Equipe Bases de Données, *Conception d'une base de données - Eléments méthodologiques*, Institut d'Informatique, FUNDP, Namur, March 1987

[Falk 89]
Falkenberg, E. D., Lindgreen, P. (eds.), *Information system concepts: an in-depth analysis*, Proceedings of the IFIP TC 8/ WG 8.1 Working Conference, Elsevier Science Publishers B. V., North Holland, 1989, 357 p.

[Fike 85]
Fikes, R., Kehler, T., "The role of frame-based representation in reasoning", *Communications of the ACM*, vol. 28, nr. 9, September 1985, pp. 904-920

[Fouc 89]
Fouche, J.-J., "NIAM, une approche originale et cohérente du système d'informations, *Journ'Almin*, FUNDP, Namur, nr. 11, June 1989, pp. 23-32

[Gall 89]
Gallaire, H., Minker, J., Nicolas, J.-M., "Logic and databases: a deductive approach" in [Mylo 89], pp. 231-247

[Hain 74]
Hainaut, J.-L., Lecharlier, B., "An extensible semantic model of data base and its data language" in *Information Processing 74*, IFIP, North-Holland, 1974, pp. 1026-1030

[Hain 81]
Hainaut, J.-L., "Theoritical and practical tools for data base design" in Zaniolo, C., Delobel, C. (eds.), *Proceedings of the 7th International Conference on Very Large Data Bases*, 1981, pp. 216-224

[Hain 85]
Hainaut, J.-L., *Introduction aux systèmes de gestion de bases de données CODASYL 71*, Institut d'Informatique, FUNDP, Namur, October 1985, 48 p.

[Hain 86a]
Hainaut, J.-L., *Conception assistée des applications informatiques - 2. Conception de la base de données*, coll. Méthode + Programmes, Masson, Paris, 1986, 224 p.

[Hain 86b]
Hainaut, J.-L., *Schémas de la base des spécifications (Deuxième version)*, Database Design Workbench Project - Specification SPEC-86/7-SPEC-86/8-4, Institut d'Informatique FUNDP, Namur, August 1986

[Hain 86c]
Hainaut, J.-L., *Introduction aux outils généraux de développement de l'atelier (Première version)*, Database Design Workbench Project - Specification SPEC-86/8-1, Institut d'Informatique FUNDP, Namur, August 1986

[Hain 88]
Hainaut, J.-L., *Introduction à la théorie relationnelle des bases de données*, Draft Course Support, Institut d'Informatique, FUNDP, Namur, March 1988

[Hain 89a]
Hainaut, J.-L., *Decomposition of irreducible data base schemata*, Research Paper, Institut d'Informatique, FUNDP, Namur, January 1989, 17 p.

[Hain 89b]
Hainaut, J.-L., *Semantic equivalence of data base schemata - A new family of formal tools*, Research Paper, Institut d'Informatique, FUNDP, Namur, June 1989, 18 p.

[Hain 89c]
Hainaut, J.-L., "A generic entity-relationship model" in [Falk 89], pp. 109-138

[Hain 89d]
Hainaut, J.-L., *Bases de données et bases de connaissances en gestion des organisations*, Syllabus of the 5th AFCET Autumn Scool of Databases, Port Barcarès, France, November 1989, 121 p.

[Halb 87]
Halbert, P. C., O'Brien, P. D., "Using types and inheritance in object-oriented languages" in [Bezi 87], pp. 20-31

[Hamm 81]
Hammer, M., McLeod, D., "Database description with SDM, a semantic database model", *ACM Transaction on Database Systems*, vol. 6, nr. 3, March 1981, pp. 351-386

[Higg 89]
Higgs, B. J., *Object-oriented database systems*, Digital Equipment Corporation, Maynard, February 1989, 30 p.

[Hull 87a]
Hull, R., "A survey of theoritical research on typed complex database objects" in [Pard 87], pp. 193-256

[Hull 87b]
Hull, R., King, R., "Semantic database modeling: survey, applications, and research issues", *ACM Computing Surveys*, vol. 18, nr. 3, September 1987, pp. 201-260

[Kern 84]
Kernighan, B. W., Ritchie, D. M., *Le langage C*, Buffenoir, T. (trad.), Masson, Paris, 1984, 213 p.

[King 84]
King, R., McLeod, D., "A unified model and methodology for conceptual database design" in [Bro 84a], pp. 313-327

[Kugl 86]
Kugler, H.-J. (ed.), *Information processing 86*, IFIP, Elsevier Science Publishers B. V., North-Holland, 1986

[Kung 90]
Kung, C., "Object subclass hierarchy in SQL: a simple approach", *Communications of the ACM*, vol. 33, nr. 7, July 1990, pp. 117-125

[LePe 88]
Le, F., Peugeot, C., "Quelques problèmes liés à l'introduction du concept de généralisation/spécialisation dans le modèle entité-relation", *Modèles et Bases de Données*, nr. 10, July 1988, pp. 3-16

[Lenz 85]
Lenzerini, M., "SERM: semantic entity-relationship model" in [Chen 85], pp. 270-277

[Lepr 86]
Leprêtre, S., *La méthode IA et son outil logiciel associé sur micro-ordinateur*, Syllabus of the AFCET Days on Database Management Systems on Micro-computers, La Rochelle, France, 1986

[Ling 85]
Ling, T. W., "A normal form for E-R diagrams" in [Chen 85], pp. 24-35

[Mann 88]
Mannino, M. V., Navathe, S. B., Efelsberg, W., "A rule-based approach for merging generalization hierarchies", *Information Systems*, vol. 13, nr. 3, 1988, pp. 257-272

[Mark 89]
Markowitz, "On the correctness of representing extended entity-relationship structure in the relational model" in *Proceedings of ACM SIGMOD conference on the management of data*, 1989, pp. 430-439

[Matt 88]
Mattos, N. M., "Abstraction concepts: the basis for data and knowledge modeling" in [Bati 88a], pp. 331-350

[McLe 80]
Mc Leod, D., Smith, J. M., "Abstraction in databases" in *Proceedings of the ACM SIGMOD conference on the management of data*, ACM, 1980, pp. 19-25

[Meye 89]
Meyer, B., *Eiffel: an introduction*, TR-E1-3/GI, Interactive Software Engineering Inc., 1989, 14 p.

[Mylo 84]
Mylopoulos, J., Levesque, H. J., "An overview of knowledge representation" in [Bro 84a], pp. 3-18

[Mylo 89]
Mylopoulos, J., Brodie, M. L. (eds.), *Readings in articial intelligence & databases*, Kaufmann, Los Altos, California, 1989, 688 p.

[Nava 86]
Navathe, S. B., Elmasri, R., Larson, J., "Integrating user views in database design", *IEEE Computer*, vol. 19, nr. 1, January 1986, pp. 50-61

[Nava 88a]
Navathe, S. B., Pillalamarri, M. K., "OOER: toward making the E-R approach object-oriented" in [Bati 88a], pp. 56-76

[Nava 88b]
Navathe, S. B., *Conceptual and logical database design*, Slides of the 7th International Conference on Entity-Relationship Approach, Rome, Italy, November 1988

[Nijs 86]
Nijssens, G. M., "On experience with large-scale teaching and use of fact-based conceptual schemas in industry and university" in Steel, T. B., Meersman, R. (eds.), *Database semantics*, DS-1 IFIP Proceedings, Elsevier, North-Holland, 1986, pp. 189-204

[Pard 87]
Paredaens, J. (ed.), *Databases*, Academic Press, London, 1987

[Pare 87]
Parent, C., Spaccapietra, S., "Un modèle et une algèbre pour les bases de données de type E/R", *Techniques et Sciences Informatiques*, vol. 6, nr. 5, pp. 435-456

[Pare 89]
Parent, C., Spaccapietra, S., "About entities, complex objects and object-oriented data models" in [Falk 89], pp. 193-224

[Pate 90]
Patel-Schneider, P. F., "Practical, object-based knowledge representation for knowledge-based systems", *Information Systems*, vol. 15, nr. 1, 1990, pp. 9-19

[Peck 88]

Peckham, J., Maryanski, F., "Semantic data models", *ACM Computing Surveys*, vol. 20, nr. 3, September 1988, pp. 153-198

[Petz 88]

Petzold, C., *Programming WINDOWS*, Microsoft Press, Washington, 1988, 852 p.

[Pion 88]

Pionnier, J., "Réflexions sur certaines impossibilités de représentation de la réalité des données par le symbolisme du modèle entité-association", *Modèles et Bases de données*, nr. 8, February 1988, pp. 31-35

[Plet 89]

Pletch, A., "Conceptual modeling in the classroom", *ACM SIGMOD RECORD*, vol. 18, nr. 1, March 1989, pp. 74-80

[Plum 86]

Plum, T., *Le langage C - Introduction à la programmation*, M. Rousseau (trad.), coll. Intermicro, Prentice-Hall/Inter-Editions, Paris, 1986, 337 p.

[Pott 89]

Potter, W. D., Trueblood, R. P., Eastman, C. M.,"Hyper-semantic data modeling", *Data & Knowledge Engineering*, vol. 4, nr. 1, July 1989, pp. 69-90

[Prov 89]

Provost, I., *Les objets interactifs*, Institut d'Informatique, FUNDP, Namur, 1989

[Qian 85]

Qian, X., Wiederhold, G., "Data definition facility of CRITIAS" in [Chen 85], pp. 46-55

[Ridj 84]

Ridjanovic, D., Brodie, M. L., "On the design and specification of database transactions" in [Bro 84a], pp. 277-306

[Roch 88]

Rochfeld, A., Morejon, J., "MERISE : Une méthode en mouvement - Extensions au modèle conceptuel des données", *Modèles et Bases de Données*, nr. 8, February 1988, pp. 37-56

[Roll 86]

Rolland, C., Proix, C., "An expert system approach to information system design" in [Kugl 86], pp. 241-250

[Roll 89]

Rolland, C., *Méthodes et outils d'aide à la conception des systèmes d'information - Evolution et perspectives*, Syllabus of the Conference on New perspectives in database, University of Geneva, September 1990

[Rous 84]
Roussopoulos, N., Yeh, R. T., "An adaptable methodology for database design", *IEEE Computer*, vol. 17, nr. 5, May 1984, pp. 64-80

[Sacc 88]
Sacco, G. M., "The FACT model: a semantic data model for complex databases", *Information Systems*, vol. 13, nr. 1, 1988, pp. 1-11

[Sacr 88]
Sacré, B., *Programmation sous WINDOWS*, Institut d'Informatique, FUNDP, Namur, April 1988

[Saka 83]
Sakai, H., "Entity-relationship approach to logical database design" in [DaJa 83], pp. 155-187

[Schi 83]
Schiel, U., "An abstract introduction to the temporal hierarchic data model (THM)" in Schkolnick, M., Thanos, C. (eds.), *Proceedings of the 9th International Conference on Very Large Data Bases*, 1988, pp. 322-330

[Sern 85]
Sernadas, A., Sernadas, C., "The use of E-R abstractions for knowledge representation", in [Chen 85], pp. 224-231

[Ship 81]
Shipman, D. W., "The functional data model and the data language DAPLEX", *ACM Transactions on Database Systems*, vol. 6, nr. 1, March 1981, pp. 140-173

[Shla 88]
Shlaer, S., Mellor, S., *Object-oriented systems analysis - Modeling the world in data*, Yourdon Press, 1988, 144 p.

[Simo 89]
Simovici, D. A., Stefanescu, D. C., "Formal semantics for database schemas", *Information Systems*, vol. 14, nr. 1, 1989, pp. 65-77

[Smit 77]
Smith, J. M., Smith, D. C. P., "Database abstraction: aggregation and generalization", *ACM Transactions on Database Systems*, vol. 2, nr. 2, June 1977, pp. 105-133

[Smit 80]
Smith, D. C. P., Smith, J. M., "Conceptual data base design" in *Infotech state of the art report on data design*, 1980

[Sowa 84]
Sowa, J. F., *Conceptual structures: information processing in mind and machine*, Addison-Wesley, Reading, Massachusetts, 1984, 469 p.

[Spac 89]
Spaccapietra, S., Parent, C., Yetognon, K., Abaidi, M. S., "Generalizations: a formal and flexible approach" in Prakash, N. (ed.), *Management of Data,* Proceedings of COMAD'89, Mc Graw Hill, 1989, pp. 100-117

[Spri 88]
Springsteel, F. N., Chuang, P.-J., "ERDDS: the intelligent E-R-based database design system" in [Bati 88a], pp. 211-230

[Ston 89]
Stonebraker, M., "Future trends in database systems", *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, nr. 9, March 1989, pp. 33-44

[Stor 88]
Storey, V. C., Goldstein, R. C., "A methodology for creating user views in database design", *ACM Transactions on Database Systems*, vol. 13, nr. 3, September 1988, pp. 305-338

[SuSY 86]
Su, S. Y. W., "Modeling integrated manufacturing data with SAM*", *IEEE Computer*, vol. 19, nr. 1, January 1986, pp. 34-49

[Tabo 88]
Tabourier, Y., "Du modèle entité-relation à un véritable réseau sémantique", *Modèles et Bases de Données*, nr. 9, June 1988, pp. 3-32

[Tard 89]
Tardieu, H., Rochfeld, A., Coletti, R., *La méthode MERISE - Tome 1 - Principes et outils*, Les Editions d'Organisation, Paris, 1989

[Teor 86]
Teorey, T. J., Yang, D., Fry, J. P., "A logical design methodology for relational databases using the extended entity-relationship model", *ACM Computing Surveys*, vol. 18, nr. 2, June 1986, pp. 197-222

[Teor 89]
Teorey, T. J., Weir, G., Bolton, D. L., Koenig, J. A., "ER model clustering as an aid for user communication and documentation in database design", *Communications of the ACM*, vol. 32, nr. 8, August 1989, pp. 975-987

[Tour 86]
Touretzki, D. S., *The mathematics of inheritance systems*, Pitman, London, 1986

[Tuch 90]
Tucherman, L., Casanova, M. A., Furtado, A. L., "The CHRIS consultant - a tool for database design and rapid prototyping", *Information Systems*, vol. 15, nr. 2, 1990, pp. 187-195

[Urba 87]
Urban, S. D., Delcambre, L. M. L., "Perspectives of a semantic schema" in *Proceedings of the 3rd international conference on data engineering*, IEEE, 1987, pp. 485-492

[Verm 83]
Vermeir, D., "Semantic hierarchies and abstractions in conceptual schemata", *Information Systems*, vol. 8, nr. 2, 1983, pp. 117-124

[Vigi 90]
Vigier, P., Zeippen, J.-M., *Conceptual and logical database design*, Draft Course Support, Digital Equipement Corporation, Maynard, April 1989

[vLam 82]
van Lansweerde, A., "Les outils d'aide au développement de logiciels - Un aperçu des tendances actuelles" in *Proceedings JIIA 82*, Paris, 1982

[vLam 88a]
van Lamsweerde, A., Delcourt, B., Delor, E., Schayes, M.-C., Champagne, R., "Generic lifecycle support in the ALMA environment", *IEEE Transactions on Software Engineering*, June 88, pp. 720-741

[vLam 88b]
van Lamsweerde, A., *Méthodologie de développement de logiciels*, Courses Notes, Institut d'Informatique, FUNDP, Namur, 1988

[vLam 88c]
van Lamsweerde, A., *Techniques d'intelligence artificielle*, Course Notes, Institut d'Informatique, FUNDP, Namur, 1988

[vLam 90]
van Lamsweerde, A., "Active software objects in a knowledge-based lifecycle support environment" to appear in Meyer, B., Mandrioli, D. (eds.), *Object-oriented programming and other advanced design techniques*, Springer-Verlag, 1990

[Wagn 88]
Wagner, C. F., "Implementing abstraction hierarchies" in [Bati 88a], pp. 267-282

[Warn 88]
Warnant, G., *Eléments de modélisation du dialogue d'une application interactive*, Research Paper, Institut d'Informatique, FUNDP, Namur, 1988

[Wied 86]
Wiederhold, G., "Views, objects and databases", *IEEE Computer*, vol. 19, nr. 12, December 1986, pp. 37-44

[Wood 75]
Woods, W. A., "What's in a link: foundations for semantic networks" in Bobrow D. G., Collins, A.M. (eds.), *Representation and understanding: studies in cognitive science*, New York, Academic Press, 1975, pp. 35-82 (also in [Brac 85] pp. 217-242)

[Your 89]
Yourdon, E., *Modern Structured Analysis*, Prentice-Hall, Englewood Cliffs, N. J., 1989, 672 p.

[Zani 83]
Zaniolo, C., "The database language GEM" in *Proceedings of the ACM SIGMOD conference on management of data*, 1983, pp. 207-217

[Zill 84]
Zilles, S. N., "Types, algebras and modelling" in [Brod 84a], pp. 441-450

Classification of references

- Artificial intelligence - Programming - Databases

  [Bane 87] [Berg 88a] [Brod 80] [Brod 84a] [Brod 86] [Chen 85] [Gall 89] [Matt 88] [Mylo 89] [Pott 89] [Sern 85] [Sowa 84] [vLam 90] [Zill 84]

  - Programming - Programming languages

    - Object-oriented programming

      [Amer 87] [Bezi 87] [Borg 88] [Duco 87] [Duco 89] [Halb 87] [Meye 89]

    - Windows programming

      [Petz 88] [Prov 89] [Sacr 88]

    - C language

      [Kern 84] [Plum 86]

  - Artificial intelligence - Knowledge representation and processing

    [Alte 90] [Brac 83] [Brac 85] [Brac 89] [Dard 90] [Fike 85] [Mylo 84] [Pate 90] [Tour 86] [vLam 88c] [Wood 75]

- Databases

  [Aria 86] [Pard 87] [Simo 89] [Wied 86]

  - DBMSs - Object-oriented DBMSs

    [Ditt 90] [Higg 89] [Ston 89]

  - Data(base) models - Semantic data models

    [Abri 74] [Afsa 86] [Batr 90] [Berm 86] [Fouc 89]
    [Gall 89] [Hain 74] [Hain 89c] [Hamm 81]
    [Hull 87a] [Hull 87b] [Lepr 86] [Pare 89]
    [Peck 88] [Qian 85] [Sacc 88] [Schi 83] [Ship 81]
    [SuYS 86]

    - Network models

      [Hain 85] [Vigi 90]

    - Relational models

      [Codd 79] [Date 86] [Hain 88]
      [Hain 89c] [Kung 90] [Vigi 90]

    - E-R approach

      [Bati 88a] [Berg 88b] [Boda 89]
      [Chen 76] [Chen 81a] [Chen 83]
      [Chen 85] [Coll 88] [Czed 90]
      [DaJa 83] [Davi 89] [DiBa 89]
      [Doga 81] [Elma 85] [Hain 89c]
      [Hain 89d] [LePe 88] [Lenz 85]
      [Ling 85] [Nava 88b] [Pare 87]
      [Pion 88] [Roch 88] [Saka 83]
      [Spac 89] [Tabo 88] [Tard 89]
      [Teor 86] [Teor 89] [Vigi 90]
      [Zani 83]

- Software development - CASE

  - Information system and development methodologies

    [Boda 89] [Bube 87] [Fouc 89] [Kugl 86] [Roll 86] [Roll 89]
    [Tard 89] [Your 89]

  - Software engineering

    [Borg 84] [DaJa 83] [vLam 88b]

  - Software engineering environments

    [vLam 82] [vLam 88a] [vLam 90]

- Database design

  [Bati 88b] [Blah 88] [Ceri 81] [Dols 88] [Equi 87] [Hain 87] [Hain 86a] [Nava 88b] [Ridj 84] [Rous 84] [Saka 83] [Stor 88] [Teor 86] [Teor 89] [Urba 87] [Verm 83] [Vigi 90]

  - Conceptual modelling - Conceptual models

    [Boda 89] [Borg 85] [Brod 80] [Brod 81] [Bube 87] [King 84] [Ling 85] [Nijs 86] [Plet 89] [Roch 88] [Shla 88] [Smit 80] [Sowa 84] [Tard 89] [Verm 83] [Your 89]

  - Schema transformation

    [Hain 81] [Hain 86a] [Hain 89a] [Hain 89b] [Hain 89d] [Mark 89]

  - Schema integration

    [Bati 86] [Cive 88] [Mann 88] [Nava 86]

- Database design workbenches

  [Bati 82] [Berm 86] [Bouz 84] [Bouz 86] [Bria 85] [Cata 88] [Cive 88] [Roll 86] [Spri 88] [Tuch 90]

  - TRAMIS

    [Conc 90a] [Conc 90b] [Hain 86b] [Hain 86c]

- Abstraction mechanisms

  [Boda 89] [Brod 80] [Brod 81] [Brod 84a] [Codd 79] [Davi 89] [Hamm 81] [Hull 87b] [Matt 88] [McLe 80] [Mylo 84] [Nava 88a] [Ridj 84] [Smit 77] [Verm 83]

  - Generalization/specialization - Inheritance

    [Amer 87] [Bana 87] [Berg 88b] [Borg 84] [Borg 88] [Brac 83] [Carbo 80] [Duco 87] [Duco 89] [Elma 85] [Halb 87] [Kung 90] [LePe 88] [Mann 88] [Nava 88a] [Smit 77] [Spac 89] [Tour 86] [Vigi 90] [Wagn 88]

Acronyms

ADL	Access algorithm Description Language
CAD/CAM	Computer-Aided Design/Computer-Aided Manufacturing
CASE	Computer-Aided Software Engineering
DBMS	Data Base Management System
DMS	Data Management System
E-R	Entity-Relationship
GAM	Generalized Access Model
GER	Generic Entity-Relationship
ISL	Information system Specification Language
UoD	Universe of Discourse
WIMP	Window Icon Mouse Pull-down menu

Appendix A

Reference example

This example has been adapted from [Peck 88] and serves as a running example throughout the text. Using a consistent example for the explanation of the concepts allows for easier understanding as examples are from a same UoD. We do not claim that this example is a comprehensive description of all aspects of a 'real world' database. Its role is essentially pedagogical: therefore only elements essentials for our illustrating purpose are presented. Some variations on this UoD are made along the different chapters.

The UoD of this example is a library. We present its description in a more or less structured, textual form: one may consider the statements above as the results of interviews and analysis of documents concerning the library.

The statements expressing UoD semantics (and assumed self-explaining) are:

- a publication has a title, a topic, an ISBN code, and is written by authors;
- authors have a name, a stipend and may write many publications;
- a journal paper is characterized by the volume and the number of the journal in which it appeared;
- books, journal papers and conference papers are particular kinds of publications;
- a book is characterized by a number of sales, a topic, and a price; it has been written by one or several authors and is published by one publishing company;
- a best-seller is a book with a number of sales larger than 10,000;
- a publishing company has a name and publishes any number of books;
- a person has a name, an address, and a posible phone number;
- a reviewer is a person;

- persons may borrow books and for each borrowing a due-date is registered;
- books are often reviewed by a reviewer (this link is characterized by a date and a rating)
- a literary figure is both an author and a reviewer;
- the set of database books contains all books with topic = *DB*, the topic of artificial intelligence books is *AI*, database books are characterized by the data model used;
- some books are both database and artificial intelligence books;
- the set of good books is a subset of books and is identified by the end-user;
- all research books are good books, but there are good books of no concern for research;
- a publication is identified by its ISBN code.

Appendix B

The GER model

In order to put in evidence the different facets of the generalization/specialization concepts, we use the *GER* (*Generic Entity-Relationship*) model as it has been designed for being a formal framework in which models based on the *object-association* philosophy, i.e. in which the real world is perceived as a collection of objects which are in association with each other, may be described, compared, and cross-translated.

The following definitions give a precise but intuitive perception of the GER model. For more information, consult [Hai 89c], from which the following text has been adapted.

B.1. Entity domains

An *entity*[1] is an element of the *universal entity domain* called ENTITIES. Entities are created and can be deleted. At any given time, all entities are distinct. A new entity domain E_2 can be defined so that each of its elements, at any time, belongs to entity domain E_1 (E_2: E_1). E_2 is called a *subdomain* of E_1 and E_1 a *superdomain* of E_2. If E_2 is explicitly declared as a subdomain of E_1, it is a *direct subdomain* of E_1, and the latter is also a *direct superdomain* of E_2. Note that if E_3 is a subdomain of E_2 and E_2 is a direct subdomain of E_1, then E_3 is a subdomain of E_1. An entity domain can be a subdomain of more than one superdomain. A direct subdomain of the universal entity domain is called a *basic entity domain*. An entity domain can be declared a subset of a constructed domain obtained by applying the traditional relational operators (powerset, union, intersection, difference and projection). Entity domains are given distinct names. At any time, an entity can enter or leave a subdomain of its basic entity domain. The latter, homewer, is unique and fixed during the life of the entity. All (past, present, future) entities that are in an entity domain are said to be of a given *entity type*. That type is defined by all the structural and behavioural properties that are common to its entities. It is given the name of its domains by which it is denoted. We must always

[1] It is used to represent a UoD object.

have in mind that an entity type includes two 'aspects': the *domain*, *class* or *set* aspect and the *type* aspect. The *type* formally defines the structure of its instances, its schema (intensional description), while the *class* is the set of all instances at a given time (extensional description). For example, schema B.1 presents two entity types (their entity domain is a basic one).

B.2. Value domains

A *value* is any permanent symbol that can be stored, transmitted and processed in a manual or automated information processing system[2]. A value has a *type* which defines the set of possible values, the properties and the processing rules of the values. A *value domain* is a named set of values of a given type. There exist predefined basic value types, namely *integer*, *real*, *character*, *string*, *date*, etc. The *basic value domain* of a basic type is the set of all the possible values of that type. A new value domain can be defined as a subset of an existing one or as a subset of a constructed value domain. The legal constructors are the relational cartesian, powerset and list constructors. A *simple value domain* is either a basic value domain or a subset of simple value domain. Constructed value domains are called *complex value domains*. Schema B.2 presents a sample of value domains useful in our reference example.

B.3. Entity relation schemata

An *entity relation schema* is the descriptive relation schema that define the *attributes* (E-R meaning) of the entity type. The described entity type is a GER attribute of the entity relation schema. That attribute is called the *described entity attribute* and is based on the described entity domain. It is the primary key of the relation schema. One suggests the specific syntax *desc-of-E* for a descriptive entity schemata of entity type E. However, more than one descriptive relation schema can be defined for each entity type (with ad-hoc naming conventions). For example, schema B.3 presents the description of a book.

[2] It is used to represent a property of the UoD.

B.4. Relationship relation schemata

A *relationship* is an aggregate of at least two entities and/or relationships, together with any number of values. Any relationship belongs to a *relationship type* that is defined by all the structural and behavioural properties that are common to its relationships. A *relationship relation schema* [3] describes the roles and the E-R attributes of the relationship type. At least two GER attributes must be defined on entity domains. Its name is that of the relationship type described. Note that *cardinality constraint* i-j of each role has been limited to the most useful values, i.e. i = 0 or 1, and j = 1 or ∞. These values can be losslessly translated into key and identity constraints (see below). Considering the cardinality constraint i-j of role r_k played by entity type E_k in relationship type R:

- i = 1 is equivalent to $E_k = R[r_k]$, and i = 0 when that constraint does not stand;
- j = 1 is equivalent to the specification of r_k as a key of R, and j = ∞ when that constraint does not stand.

For example, schema B.4 expresses that authors write at least one publication and that a publication is written by one or several authors.

B.5. Constraints

B.5.1. Key constraints

Any entity and relationship relation schema has at least one *key*. A component of a key is an attribute, or an attribute of a cartesian attribute, or an element of a powerset attribute. When possible, a key is specified by the underlined notation (the ISBN_CODE in schema B.3, for example).

B.5.2. Inclusion and identity constraints

One can define *inclusion* (⊆) and *identity* (=) *constraints* between any two entity sets, value set or relations (see schema B.4). Traditional

[3] We do not use the concise notation for functional relationship types proposed in [Hain 89c].

relational notations can be used. Keep in mind that:

- entity sets are entity domains or projections of entity relation schemata, of relationship relation schemata or of algebraically constructed relation schemata;
- value sets are defined by projection of entity relation schemata, of relationship relation schemata or of algebraically constructed relation schemata;
- relations are entity relation schemata, relationship relation schemata or algebraically constructed relation schemata.

Appendix C

Extract of TRAMIS user interface monitoring

This appendix explains the monitoring of TRAMIS user interface (version 1.01) by showing how the different interactive objects (Windows approach) appear during a working session. Actually, the current interface is in French but we took the liberty to translate it in English. Moreover only the aspects which are relevant will be presented (namely those concerning conceptual aspects). It is more especially intended for the readers acquainted with TRAMIS user interface.

GLOBAL window

GLOBAL menu

...

Elaborate	Detailed design of the current schema ==> ENTITY window
Transform	
Global	Global transformations of the current schema
Toward models	Transformations of the current schema towards a model
Produce	
Conformity	Model compliance checking
Generation	Production of external descriptions
Report	
Complete	Production of a detailed report on the complete schema

Partial	Production of a detailed report on an object
Global	Production of a global report on the current schema
Schema	Consultation and modification of schemata
Load	Loading of external descriptions

ENTITY window

ENTITY Menu

Consult

Complete	Choice of the ATTRIBUTE, RELATIONSHIP and GROUP subwindows for the presentation of the current entity type (by default)
Structure	Choice of the ATTRIBUTE subwindow
Neighbouring	Choice of the RELATIONSHIP subwindow
Groups	Choice of the GROUP subwindow
Description	Consultation of the description of the current entity type
Note	Consultation of the technical note of the current entity type
Static statistics	Consultation of the static statistics of the current entity type
...	

Modify

Characteristics	Modification of the characteristics of the current entity type ==> MODIFY_ENTITY_CHARACTERISTICS dialogue box
Description	Modification of the description of the current entity type

Note	Modification of the technical note of the current entity type
...	

Create	
Entity type	Creation of a new entity type
Relationship type	Creation of a new relationship type
...	
Delete	Deletion of the current entity type
Quit	Return to the global menu and window ==> GLOBAL window

MODIFY_ENTITY_CHARACTERISTICS Dialogue box

Fields

Name	Name of the current entity type
Short name	Short name of the current entity type
Date	Date of the current entity type
Origin	Origin of the current entity type
Population	Average size of the population of the current entity type

Buttons

OK	Modification of the current entity type ==> ENTITY Menu or Exception
Cancel	Cancelling of the modification ==> ENTITY Menu

Exceptions

Name invalid or absent!
Short name invalid or absent!
Name already used!
Short name already used!
Invalid date!
Invalid population average size!

ENTITY Active texts

The name of a relationship type in the RELATIONSHIP subwindow
==> This relationship type becomes current and the RELATIONSHIP window is presented

The name of an entity type in the RELATIONSHIP subwindow
==> This entity type becomes current and the RELATIONSHIP window is presented

The name of an attribute in the ATTRIBUTE subwindow
==> This attribute becomes current and the ATTRIBUTE window is presented

Any area of the ATTRIBUTE subwindow
==> Detailed design of an attribute - the ATTRIBUTE window is presented

The name of a group in the GROUP subwindow
==> This group becomes the current and the GROUP window is presented

Any area of the GROUP subwindow
==> Detailed design of a group - the GROUP window is presented

The name of a component in the GROUP subwindow
==> This component becomes current and the COMPONENT window is presented

Appendix D

Extract of TRAMIS user interface monitoring including the category

---

This appendix explains the modifications of the monitoring of TRAMIS user interface (see appendix C) when the proposed category construct is introduced.

GLOBAL window

GLOBAL menu

...

Elaborate	Detailed design of the current schema ==> ENTITY window
Transform	
Global	Global transformations of the current schema[1]
Toward models	Transformations of the current schema towards a model
Produce	
Conformity	Model compliance checking
Generation	Production of external descriptions[2]
Report	
Complete	Production of a detailed report on the complete schema[3]

---

[1] Three new transformations are proposed for the elimination of categories from a schema.
[2] The category is another aspect proposed for the generation (of ISL descriptions).
[3] The user is asked for the application of 'automatic' downward inheritance in the report.

Partial	Production of a detailed report on object[4]
Global	Production of a global report on the current schema[5]
Schema	Consultation and modification of schemata[6]
Load	Loading of external descriptions[7]

ENTITY window

ENTITY Menu

Consult

Complete	Choice of the ATTRIBUTE, RELATIONSHIP and GROUP subwindows for the presentation of the current entity type (by default)
Structure	Choice of the ATTRIBUTE subwindow
Neighbouring	Choice of the RELATIONSHIP subwindow[8]
Groups	Choice of the GROUP subwindow
Description	Consultation of the description of the current entity type
Note	Consultation of the technical note of the current entity type
Static statistics	Consultation of the static statistics of the current entity type
...	
Inheritance	'Automatic' application of downward inheritance[9]

[4] The user is asked for the application of 'automatic' downward inheritance in the report.
[5] The report contains the number of categories of the schema (if any).
[6] The summarized statistics contains the number of categories of the schema (if any).
[7] The category is another aspect proposed for the loading (of ISL descriptions).
[8] This subwindow also presents the (direct and indirect) generic entity types, and the (direct) specific entity types of the current entity type, with the class inclusion constraints.
[9] If this item is chosen, then the inherited attributes, roles, relationship types and groups are presented in the ATTRIBUTE, RELATIONSHIP, and GROUP subwindows respectively. It is available only if the current entity type is in an *is-a* hierarchy.

Modify	
Characteristics	Modification of the characteristics of the current entity type ==> MODIFY_ENTITY_CHARACTERISTICS dialogue box
Description	Modification of the description of the current entity type
Note	Modification of the technical note of the current entity type
...	
Create	
Entity type	Creation of a new entity type
Relationship type	Creation of a new relationship type
...	
Delete	Deletion of the current entity type
Transform	Elementary ransformations concerning the category[10]
New generic	Transformation of point 6.3.1.G.a
New partition	Transformation of point 6.3.1.G.e
New specific	Transformation of point 6.3.1.G.g
Covering	Transformation of point 6.3.1.G.c
Disjunction	Transformation of point 6.3.1.G.d
In generic	Transformation of point 6.3.1.G.h
By specific	Transformation of point 6.3.1.G.b
By generic	Transformation of point 6.3.1.G.f
By relationship types	Transformation of point 6.3.1.G.i
Quit	Return to the global menu and window ==> GLOBAL window

MODIFY_ENTITY_CHARACTERISTICS Dialogue box

Fields

Name	Name of the current entity type
Short name	Short name of the current entity type

[10] This menu item is available only if the current entity type is in an is-a hierarchy.

Date	Date of the current entity type
Origin	Origin of the current entity type
Population	Average size of the population of the current entity type

Buttons

OK	Modification of the current entity type ==> ENTITY Menu or Exception
Cancel	Cancelling of the modification ==> ENTITY Menu
Is-a	Modification of generalization/specialization characteristics of current entity type ==> MODIFY_IS_A_CHARACTERISTICS dialogue box

Exceptions

Name invalid or absent!
Short name invalid or absent!
Name already used!
Short name already used!
Invalid date!
Invalid population average size![11]

MODIFY_IS_A_CHARACTERISTICS Dialogue box

Fields

Generic	Name of the generic entity type (if any)

List boxes

Specific	List of the names of the specific entity types
New specific	List of the name of possible new specific entity types

[11] A population average size is also invalid if it does not verify the statistical equations.

New generic	List of the name of possible new generic entity types

Buttons

Remove generic	Removal of the generic entity type
Remove specific	Removal of a specific entity type
Add generic	Adding of a new generic entity type
Add specific	Adding of a new specific entity type
Covering	Adding of the covering constraint
Disjunction	Adding of the disjunction constraint
OK	Modification of the current entity type ==> MODIFY_ENTITY_CHARACTERISTICS dialogue box or Exception
Cancel	Cancelling of the modification ==> MODIFY_ENTITY_CHARACTERISTICS dialogue box

Exceptions

Statistical equations are not verified !

ENTITY Active texts[12]

The name of a relationship type in the RELATIONSHIP subwindow
==> This relationship type becomes current and the RELATIONSHIP window is presented

The name of an entity type in the RELATIONSHIP subwindow
==> This entity type becomes current and the RELATIONSHIP window is presented

The name of an attribute in the ATTRIBUTE subwindow
==> This attribute becomes current and the ATTRIBUTE window is presented

[12] When the automatic inheritance button is ON, the different subwindows contain the name of the generic entity types from which the 'characteristics' are inherited. These names are active texts too.

Any area of the ATTRIBUTE subwindow

==> Detailed design of an attribute - the ATTRIBUTE window is presented

The name of a group in the GROUP subwindow

==> This group becomes the current and the GROUP window is presented

Any area of the GROUP subwindow

==> Detailed design of a group - the GROUP window is presented

The name of a component in the GROUP subwindow

==> This component becomes current and COMPONENT window is presented

Appendix E

ADL algorithms concerning the category

Actually, this appendix describes an example of the ADL algorithms for the functions specified in chapter 6. The addendum contains a more complete description of these algorithms. They are working on the GAM schema of the specification database (figure E.1). We use ADL primitives consistent [Hain 86a] with the access module of TRAMIS.

Algorithm E.1

```
procedure CreateIsa (var Es, Eg: record-var);

{
Specif.:

This algorithm implements the function specified[1] in 6.3.1.B.a to the
ENTITY_Ts Es (the specific entity type) and Eg (the generic entity type)
which are assumed verifying its precondition.
}

begin

  Cat:= G_OF_SUBTYPE(GST_E: Eg);
  if Cat = ()  then
    create Cat:= G_OF_SUBTYPE((: COVERING = false)
                              and (: EXCLUSIVE = false))
  endif;
  create Subt:= SUBTYPE((ST_E: Es) and (ST_GST: Cat));
  if COVERING(: Cat) and EXCLUSIVE(: Cat) then
    {
    Apply the statistical inference rules for the category of Eg
    }
    StatInfer(Eg)
  endif;

end;
```

---

[1] When the specifications of chapter 6 are sufficient for a *precise* understanding, i.e. when the 'translation' in terms of the GAM schema is obvious, we do not provide a new specification in GAM terms.

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Generalization/specialization abstraction structures

-

Theoretical study and integration in a database design workbench

Jean-Marc ZEIPPEN

- Addendum -

Mémoire présenté sous la direction du Professeur Jean-Luc HAINAUT pour l'obtention du titre de Licencié et Maître en Informatique

Année académique 1989-1990

Addendum

A more complete description of the ADL algorithms concerning the category

---

This addendum completes the appendix E. It presents the main interesting ADL algorithms (and their specifications[1]) for the functions specified in chapter 6, i.e. those concerning 'directly' the categories and presenting a few difficulties. These algorithms are working on the GAM schema of the specification database (see figure E.1, page Q). They are consistent with [Hain 86a] the access module of TRAMIS.

Algorithms for the modifications

Algorithm E.1

```
procedure CreateIsa (var Es, Eg: record-var);
{
Specification:

▪ Precondition:
                          • ENTITY_T Es does not belong to a category
                          • Eg is not a (direct or indirect) specific ENTITY_T
                            of Es
                          • Statistical equations are verified (when
                            considering that Es is-a Eg)
▪ Effect:
                          • This algorithm implements the function specified
                            in 6.3.1.B.a (p. 81) to the ENTITY_Ts Es and Eg
}
begin
  Cat:= G_OF_SUBTYPE(GST_E: Eg);
  if Cat = ()  then
    create Cat:= G_OF_SUBTYPE((: COVERING = false)
                              and (: EXCLUSIVE = false));
  endif;
  create Subt:= SUBTYPE((ST_E: Es) and (ST_GST: Cat));
```

---

[1] When the specifications of chapter 6 are sufficient for a *precise* understanding, we keep them in natural language in order to make the reading easier.

```
  if (COVERING(: Cat) and EXCLUSIVE(: Cat)) then
    {
    Apply the statistical inference rules for the category of Eg
    }
    StatInfer(Eg);
  endif;
end;
```

Algorithm E.2

```
procedure ModifyCic (var Eg: record-var; Cover, Disj: boolean);
{
Specification:

▪ Precondition:
                          • Eg is an ENTITY_T with a category Cat
                          • Statistical equations are always verified when
                            EXCLUSIVE(: Cat) = Disj and COVERING(: Cat)
                            = Cover
▪ Effect:
                          • This algorithm implements the function specified
                            in 6.3.1.B.c (p. 82) for Cat
}
begin
  Cat:= G_OF_SUBTYPE(GST_E: Eg);
  modify Cat((: COVERING = Cover) and (: EXCLUSIVE = Disj));
  if Cover and Disj then
    {
    Apply the statistical inference rules for the category of Eg
    }
    StatInfer(Eg);
  endif;
end;
```

Algorithm E.3

```
procedure SuppressIsa (var Es, Eg: record-var);
{
Specification:

▪ Precondition:
                          • Es is a direct specific ENTITY_T of Eg (the
                            generic ENTITY_T)
▪ Effect:
                          • This algorithm implements the function specified
                            in 6.3.1.B.b (p. 82) for Es and Eg
}
begin
  {
  Suppress the is-a relation and suppress the category if Es is the sole
  specific entity type
  }
  Subt:= SUBTYPE((ST_E: Es));
  delete Subt;
  {
  Suppress inherited roles/attributes from groups of Es and of its
```

```
specific entity types if any
}
newlist(ListSpec); { This list contains the ENTITY_Ts to check for the
                presence of attributes/roles inherited from
                     Eg in their group }[2]
newlist(ListGen); { This list contains the ENTITY_Ts from which
                                roles/attributes can always be inherited }
addlist(ListSpec, Es);
E:= Es;
while E <> () do
  removelist(ListSpec, E);
  addlist(ListGen, E);
  {
  Verify the groups of E
  }
  Gr:= first(ID_KEY_ORD(IKO_EL: E));
  while Gr <> () do
    Co:= first(COMPONENT(C_IKO: Gr));
    while Co <> () do
      if TYPE_O(: Co) = 'Attr' then
        At:= ATTRIBUTE(RA_C: Co);
        Ent:= ENTITY_T(EL_AT: At);
        if not inlist(ListGen, Ent) then
          delete Co;
        endif;
      else { TYPE_O = 'Role' }
        Ro:= ROLE(RA_C: Co);
        Ero:= E_ROLE(ER_R: Ro);
        Ent:= ENTITY_T(E_ER: Ero);
        if not inlist(ListGen, Ent) then
          delete Co;
        endif;
      endif;
      Co:= next(COMPONENT(C_IKO: Gr));
    endwhile
    Gr:= next(ID_KEY_ORD(IKO_EL: E));
  endwhile;
    {
    Find the specific ENTITY_Ts of E
    }
  Cat:= G_OF_SUBTYPE(GST_E: E);
  if Cat <> () then
    Subt:= first(SUBTYPE(ST_GST: Cat));
    while Subt <> () do
      Ent:= ENTITY_T(E_ST: Subt);
      addlist(ListSpec, Ent);
      Subt:= next(SUBTYPE(ST_GST: Cat));
    endwhile
  endif;
  {
```

[2] We assume that the following functions for the management of lists are predefined:

- newlist(L): init an empty list L;
- addlist(L, E): add the element E to the list L;
- removelist(L, E): remove the element E from the list L;
- inlist(L, E): returns *true* if E is in the list L, *false* otherwise;
- firstlist(L): returns the first element of the list L;
- nextlist(L) : returns the element following (the last obtained element) of the list L;
- copylist(L1, L2): copy the list L2 in the list L1.

```
    Next ENTITY_T to be checked
    }
    E:= firstlist(ListSpec);
  endwhile;
end;
```

Algorithms for the transformations

Algorithm E.4

```
procedure NewGenericET (var ListSpec: list-record-var
                        Name: char(30);
                        ShortName: char(7));
{
Specification:

▪ Precondition:
                        • ListSpec is a (non empty) list of ENTITY_Ts of a
                          same SCHEMA which do not belong to a category
                        • Name (resp. ShortName) is not the name (resp.
                          short name) of an ENTITY_T of the schema
▪ Effect:
                        • This algorithm implements the transformation
                          specified in 6.3.1.G.a (p. 86)
}
begin
  {
  Create the generic entity type and its category
  }
  Sch:= SCHEMA(S_E: firstlist(ListSpec));
  create Eg:= ENTITY_T((: E_NAME = Name)
                       and (: SHORTNAME = ShortName)
                       and (E_S: Sch));
  create Cat:= G_OF_SUBTYPE((: COVERING = true) and (: EXCLUSIVE = true)
                            and GST_E: Eg));
  Es:= firstlist(ListSpec);
  while Es <> () do
    create Subt:= SUBTYPE((ST_GST: Cat) and (ST_E: Es));
    Es:= nextlist(ListSpec);
  endwhile;
  {
  Apply the statistical inference rules for the category of Eg
  }
  StatInfer(Eg);
  {
  Update the technical note of Eg
  }
  Sys:= SYSTEM(SYS_SCH: Sch);
  create Descr:= DESCRIPTION((DESCR_SYS: Sys)
                       and (: TYPE = 'Techn_n')
                       and (: TEXT = Text[3]));
  create DesrOf:= DESCR_OF((DO_DESCR: Descr) and (DO_OBJ: Eg));
  {
```

[3] The value of *Text* is: 'This entity type results from the grouping of its specific entity types.'

```
  Upward inheritance of common attributes
  }
  ObtainCommonAttributes(ListAttr, AttrName, AnotherOne)[4];
  while  AnotherOne do
      CommonAttr:= first(ListAttr);
      modify CommonAttr((: AT_NAME = AttrName) and (AT_EL: Eg));
      removelist(ListAttr, CommonAttr);
      OtherAttr:= firstlist(ListAttr);
      while OtherAttr <> () do
        {
        Transfer the description/technical note of OtherAttr in that of
        CommonAttr } TransferDescTechOtherAttr; {
        }
        {
        delete OtherAttr;
        OtherAttr:= firstlist(ListAttr);
      endwhile;
    ObtainCommonAttributes(ListAttr, AttrName, AnotherOne);
  endwhile
  {
  Upward inheritance of common relationship types
  }
  ObtainCommonRTs(ListRT, RTName, RTShortName, AnotherOne)[5];
  while AnotherOne do
      CommonRT:= firstlist(ListRT);
      modify CommonRT((: L_NAME = AttrName)
                        and (: SHORT_NAME = RTShortName));
      Ro:= first(ROLE(R_L: CommonRT));
      while Ro <> () do
        Ero:= E_ROLE(ER_R: Ro);
        Ent:= ENTITY_T(E_ER: Ero);
        if Ent inlist(ListSpec) then
          modify Ero(ER_E: Eg);
        endif;
        Ro:= next(ROLE(R_L: CommonRT));
      endwhile;
      removelist(ListRT, CommonRT);
      OtherRT:= firstlist(ListRT);
      while OtherRT <> () do
        {
        Transfer the description/technical note of OtherRT in that of
        CommonRT } TransferDescTechNOtherRT; {
        }
        {
        delete OtherRT;
        OtherRT:= firstlist(ListRT);
      endwhile;
    ObtainCommonRTs(ListRT, RTName, RTShortName, AnotherOne);
  endwhile
end;
```

[4] If AnotherOne then ListAttr contains the list of *common* attributes (proposed by the user) to be inherited by Eg in the attribute of name AttrName (not present in Eg); ListAttr is empty otherwise.

[5] If AnotherOne then ListRT contains the list of *common* relationship types (proposed by the user) to be inherited by Eg with the name RTName (not present as a relationship type name of the schema) and the short name RTName (not present as a relationship type short name of the schema); ListRT is empty otherwise. It is assumed that the roles of these relationship types are not component of groups (simplification).

Algorithm E.5

```
procedure RepresSpecificET (var Eg: record-var);
{
Specification:

▪ Precondition:
                          • Eg has a category forming a partition
                          • Eg has no group (simplification)
                          • The average population sizes of Eg and of and its
                            specific entity types are all specified or none of
                           them are specified
▪ Effect:
                          • This algorithm implements the transformation
                            specified in 6.3.1.G.b (pp. 86-87)
}
begin
  Sch:= SCHEMA(S_E: Eg);
  Sys:= SYSTEM(SYS_SCH: Sch);
  Cat:= G_OF_SUBTYPE(GST_E: Eg);
  {
  Copy of the attributes of Eg in each specific ENTITY_T and creation of a
  work list containing these ENTITY_Ts
  }
  newlist(ListSpec);
  Subt:= first(SUBTYPE(ST_GST: Cat));
  while Subt <> () do
    Es:= ENTITY_T(E_ST: Subt);
    addlist(ListSpec, Es);
    Att:= first(ATTRIBUTE(AT_EL: Eg));
    while Att <> ()
      CopyAttr(Att, Es);
      Att:= next(ATTRIBUTE(AT_EL: Eg));
    endwhile;
    Subt:= next(SUBTYPE(ST_GST: Cat));
  endwhile;
  {
  Copy of the roles and relationship types of Eg in the specific ENTITY_Ts
  of ListSpec
  }
  Ero:= first(E_ROLE(ER_E: Eg);
  while Ero <> () do
    Ro:= ROLE(R_ER: Ero);
    Li:= LINK(L_R: Ro);
    CopyLink(Li, Ro, ListSpec);
    {
    Delete the copied LINK
    }
    delete Li;
    Ero:= first(E_ROLE(ER_E: Eg);
  endwhile
  {
  Copy of the description/technical note of Eg in each specific ENTITY_T
  and deletion of it
  }
  FindDescrTechN(Eg, Descr, TechN);
  Es:= firstlist(ListSpec);
```

```
  while Es <> () then
    FindDescrTechN(Es, DescrEs, TechNEs);
    if Descr <> () then
      if DescrEs <> () then
        modify DescrEs(: TEXT = Text[6]);
      else
        create DescrEs:= DESCRIPTION((: TYPE = 'Descr')
                                                    and (DESCR_SYS: Sys)
                                                    and (: TEXT = Text[7]));
        create DescrOf:= DESCR_OF((DO_DESCR: DescrEs)
                                                  and (DO_OBJ: Es));
      endif;
    endif;
    if TechN <> () then
      if TechNEs <> () then
        modify DescrEs(: TEXT = Text[8]);
      else
        create TechNEs:= DESCRIPTION((: TYPE = 'Tech_n')
                                                    and (DESCR_SYS: Sys)
                                                    and (: TEXT = Text[9]));
        create DescrOf:= DESCR_OF((DO_DESCR: TechNEs)
                                                  and (DO_OBJ: Es));
      endif;
    endif;
    Es:= nextlist(ListSpec);
  endwhile;
  if Descr <> () then
    delete Descr;
  endif;
  if TechN <> () then
    delete TechN;
  endif;
  {
  Replace Eg by its specific ENTITY_Ts in the category to which it belongs
  if any
  }
  SubtEg:= SUBTYPE(E_ST: Eg)
  if SubtEg <> () then
    CatEg:= G_OF_SUBTYPE(GST_ST: SubtEg);
    Es:= firstlist(ListSpec);
    while Es <> () do
      create SubtEs:= SUBTYPE((ST_GST: CatEg) and (ST_E: Es));
        Es:= nextlist(ListSpec);
    endwhile
  endif;
  {
  Delete Eg and its 'characteritics'
  }
  delete Eg;
end;
```

[6] The value of *Text* is: TEXT(: DescrEs) + 'Description of ' + E_NAME(: Eg) + ':' + TEXT(: Descr)

[7] The value of *Text* is: 'Description of ' + E_NAME(: Eg) + ':' + TEXT(: Descr)

[8] The value of *Text* is: TEXT(: TechNEs) + 'Technical note of ' + E_NAME(: Eg) + ':' + TEXT(: TechN)

[9] The value of *Text* is: 'Technical note of ' + E_NAME(: Eg) + ':' + TEXT(: TechN)

Algorithm E.6

```
procedure CoveringCategory (var Eg: record-var);
{
Specification:

▪ Precondition:
                          • Eg has a category for which the covering
                            constraint is not specified
▪ Effect:
                          • This algorithm implements the transformation
                            specified in 6.3.1.G.c (pp. 87-88)
}
begin
  Sch:= SCHEMA(S_E: Eg);
  Sys:= SYSTEM(SYS_SCH: Sch);
  Cat:= G_OF_SUBTYPE(GST_E: Eg);
  modify Cat(: COVERING = true);
  DefaultETName('OTHER' + E_NAME(: Eg), Name)[10];
  DefaultETShortName('OTHER' + SHORT_NAME(: Eg), ShortName)[11];
  create OtherEg:= ENTITY_T ((E_S: Sch)
                         and (: E_NAME = Name)
                         and (: SHORT_NAME = ShortName));
  create Subt:= SUBTYPE ((ST_E: OtherEg) and (ST_GST: Cat));
  {
  Update the origin of OtherEg
  }
  create Orig:= PROPERTY((PROP_O: OtherEg)
                                    and (: P_ROLE = 'Origin')
                                     and (: P_VALUE = E_NAME(: Eg))
                                     and (: TYPE = 'ENTITY_T'));
  {
  Update the technical note of OtherEg
  }
  create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                and (DESCR_SYS: Sys)
                             and (: TEXT = Text[12]));
  create DescrOf:= DESCR_OF((DO_DESCR: TechNEs)
                                                    and (DO_OBJ: OtherEg));
  {
  Apply statistical inference rules
  }
  if (COVERING(: Cat) and EXCLUSIVE(:Cat)) then
    StatInfer(Eg);
  endif;
end;
```

[10] DefaultETName(N1, N2) returns in N2 the value of N1 if N1 is a possible name for an ENTITY_T of the SCHEMA; otherwise, N2 contains a 'correct' name proposed by the user.

[11] DefaultETShortName(N1, N2) returns in N2 the value of N1 if N1 is a possible short name for an ENTITY_T of the SCHEMA; otherwise, N2 contains a 'correct' short name proposed by the user.

[12] The value of *Text* is: 'This entity type has been introduced to cover ' + E_NAME(: Eg) + '.'

Algorithm E.7

```
procedure DisjCategory (var Eg: record-var);
{
Specification:

• Precondition:
                        • The ENTITY_T Eg has a category which does not
                        include the
                          disjunction constraint
                        • The average population sizes of its direct
                          specific ENTITY_Ts are not specified
                        • The direct specific ENTITY_Ts play no role
                          and do not have any group (simplification)
• Effect:
                        • This algorithm implements the transformation
                          specified in 6.3.1.G.d (pp. 88-89)
}
begin
  Sch:= SCHEMA(S_E: Eg);
  Sys:= SYSTEM(DESCR_SYS: Sch);
  Cat:= G_OF_SUBTYPE(GST_E: Eg);
  modify Cat(: EXCLUSIVE = true);
  newlist(InterList);
  Subt:= first(SUBTYPE(ST_GST: Cat);
  while Subt <> () do
    Es:= ENTITY_T(E_ST: Subt);
    E:= firstlist(InterList);
    newlist(InterListBis);
    while E <> () do
      {
      Create the ENTITY_T representing the 'intersection' of Es and E (note
      that the latter represents another intersection)
      }
      DefaultETName(SHORT_NAME(: Es) + SHORT_NAME(: E), Name);
      DefaultETShortName(SHORT_NAME(: Es)[1..2]
                         + SHORT_NAME(: E)[1..2], ShortName);
      create NewE:= ENTITY_T ((E_S: Sch)
                        and (:E_NAME = Name)
                          and (:SHORTNAME = ShortName));
      create Subt:= SUBTYPE ((ST_E: NewE) and (ST_GST: Cat));
      {
      Update the origin of NewE
      }
      create Orig:= PROPERTY((PROP_O: NewE)
                                    and (: P_ROLE = 'Origin')
                                     and (: P_VALUE = E_NAME(: Eg))
                                     and (: TYPE = 'ENTITY_T'));
      {
      Update the technical note of NewE
      }
      create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                                    and (DESCR_SYS: Sys)
                                                    and (: TEXT = Text[13]);
```

[13] The value of *Text* is: 'This entity type has been introduced to have disjoint specific entity types of ' + E_NAME(: Eg) + '. It represents the intersection of ' + E_NAME(: Es) + ' and ' + E_NAME(: E) + '.'

```
      create DescrOf:= DESCR_OF((DO_DESCR: TechN)
                                and (DO_OBJ: NewE));
      {
      Copy of the attributes of E in NewE
      }
      Att:= first(ATTRIBUTE(AT_EL: E));
      while Att <> ()
        CopyAttr(Att, NewE);
        Att:= next(ATTRIBUTE(AT_EL: E));
      endwhile;
      {
      Copy of the attributes of Es in NewE
      }
      Att:= first(ATTRIBUTE(AT_EL: Es));
      while Att <> ()
        CopyAttr(Att, NewE);
        Att:= next(ATTRIBUTE(AT_EL: Es));
      endwhile;
      {
      }
      addlist(InterListBis, NewE);
      E:= nextlist(Interlist);
    endwhile
    addlist(InterList, Es);
    copylist(InterList, InterListBis);
    Subt:= next(SUBTYPE(ST_GST: Cat));
  endwhile
end;
```

Algorithm E.8

```
procedure PartitionET (var E: record-var;
             var ListAttr, ListRole: list-record-var);
{
Specification:

▪ Precondition:
                        • E does not have a category
                        • ListAttr is the set of optional attributes
                        • ListRole is the set of optional roles

▪ Effect:
                        This algorithm implements the function specified in
                        6.3.1.G.e (p. 89)
}
begin
  Sch:= SCHEMA(S_E: E);
  Sys:= SYSTEM(SYS_SCH: Sch);
  DefaultETName(E_NAME(: E) + '1', Name);
  DefaultETShortName(SHORTNAME(: E) + '1', ShortName);
  create E1:= ENTITY_T ((E_S: Sch)
                        and (: E_NAME = Name)
                        and (: SHORT_NAME = ShortName));
  DefaultETName(E_NAME(: E) + '2', Name);
  DefaultETShortName(SHORTNAME(: E) + '2', ShortName);
  create E2:= ENTITY_T ((E_S: Sch)
                         and (: E_NAME = Name)
```

```
                              and (: SHORT_NAME = ShortName));
create Cat:= G_OF_SUBTYPE((GST_E: E) and (:COVERING = true)
                                        and (:EXCLUSIVE = true));
create Subt:= SUBTYPE ((ST_E: E1) and (ST_GST: Cat));
create Subt:= SUBTYPE ((ST_E: E2) and (ST_GST: Cat));
{
Transfer the optional attributes which become mandatory in E1
}
Pos:= 1;
Att:= firstlist(ListAttr);
while Attr <> () do
  modify Attr((AT_EL: E1) and (: MIN_REP = 1) and (: POSITION = Pos));
  Pos:= Pos + 1;
  Att:= nextlist(ListAttr);
endwhile
{
Transfer the optional roles which become mandatory in E1 (their AVG_CON
is supposed ≥ 1)
}
Ro:= firstlist(ListRole);
while Ro <> () do
  modify Ro(: MIN_CON = 1));
  Ero:= E_ROLE(ER_R: Ro);
  modify Ero(ER_E: E1);
  Ro:= nextlist(ListRole);
endwhile
{
Update the origin of E1
}
create Orig:= PROPERTY((PROP_O: E1)
                                    and (: P_ROLE = 'Origin')
                                     and (: P_VALUE = E_NAME(: E))
                                     and (: TYPE = 'ENTITY_T'));
{
Update the technical note of E1
}
create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                   and (DESCR_SYS: Sys)
                                   and (: TEXT = Text[14]));
create DescrOf:= DESCR_OF((DO_DESCR: TechN)
                                                and (DO_OBJ: E1));
{
Update the origin of E2
}
create Orig:= PROPERTY((PROP_O: E2)
                                    and (: P_ROLE = 'Origin')
                                     and (: P_VALUE = E_NAME(: E))
                                     and (: TYPE = 'ENTITY_T'));
{
Update the technical note of E2
}
create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                   and (DESCR_SYS: Sys)
                                   and (: TEXT = Text[15]));
```

[14] The value of *Text* is: E_NAME(: E1) + ' contains the entities of ' + E_NAME(: E) + ' having necessarily a value for the attributes/roles contained now in this entity type.'

[15] The value of *Text* is: E_NAME(: E2) + ' contains the entities of ' + E_NAME(: E) + ' with no value for the attributes/roles contained now in ' + E_NAME(: E1) + '.'

```
  create DescrOf:= DESCR_OF((DO_DESCR: TechN)
                                    and (DO_OBJ: E2));
end;
```

Algorithm E.9

```
procedure RepresGenericET (var Eg: record-var);
{
Specification:

▪ Precondition:
                        • Eg has a category, the specific entity types of
                          which possess at the most one mandatory role or
                          only attributes and do not belong to any category
                          nor possess identifier groups
                        • Either the average population sizes of Eg and its
                          specific entity types are specified or none of
                          them
Effect:
                        • This algorithm implements the function specified
                          in 6.3.1.G.f (pp. 89-90)
}
begin
  Sch:= SCHEMA(S_E: Eg);
  Sys:= SYSTEM(SYS_SCH: Sch);
  Cat:= G_OF_SUBTYPE(GST_E: Eg);
  {
  Update the technical note of Eg
  }
  FindDescrTechN(Eg, Descr, TechN);
  if TechN <> () then
    modify TechN(: TEXT = TEXT(: TechN) + Text[16]);
  else
    create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                                  and (DESCR_SYS: Sys)
                                                   and (: TEXT = Text[17]));
    create DescrOf:= DESCR_OF((DO_DESCR: TechN)
                                                   and (DO_OBJ: Eg));
  if Descr <> () then
    modify Descr(: TEXT = TEXT(: Descr) + Text[18]);
  else
    create Descr:= DESCRIPTION((: TYPE = 'Descr')
                                                  and (DESCR_SYS: Sys)
                                                   and (: TEXT = Text[19]));
    create DescrOf:= DESCR_OF((DO_DESCR: Descr)
                                                   and (DO_OBJ: Eg));
  endif;
  {
  }
  Cov:= COVERING(: Cat);
```

[16] The value of *Text* is: 'The specific entity types have been represented by the generic entity type.'
[17] The value of *Text* is: 'The specific entity types have been represented by the generic entity type.'
[18] The value of *Text* is: 'The descriptions of the specific entity types (which have been represented by the generic entity type) follow.'
[19] The value of *Text* is: 'The descriptions of the specific entity types (which have been represented by the generic entity type) follow.'

```
Disj:= EXCLUSIVE(: Cat);
Subt:= first(SUBTYPE(ST_GST: Cat);
newlist(ListAttr); { List of the attributes representing the specific
                     entity types}
newlist(ListRole); { List of the roes representing the specific
                     entity types}
while Subt <> () do
  Es:= ENTITY_T(E_ST: Subt);
  if POPULATION(: Es) <> NULL then { POPULATION(: Eg) is also specified }
    AvgRep:= POPULATION(: Es) / POPULATION (: Eg);
  endif;
  Ero:= E_ROLE(ER_E: Es);
  if Ero <> () then
    {
    This specific ENTITY_T has only a mandatory role which is put in Eg
    }
    Ro:= ROLE(R_ER: Ero);
    addlist(listRole, Ro);
    modify Ero(ER_E: Eg);
    DefaultRoleName(Ro, E_NAME(: Es), Name)[20];
    modify Ro((: MIN_CON = 0) and (: R_NAME = Name)
              and (: AVG_CON = AvgRep));
    Li:= LINK(L_R: Ro);
    FindDescrTechN(Li, Descr, TechN);
    if TechN <> () then
      modify TechN(: TEXT = TEXT(: TechN) + Text[21];
    else
      create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                                  and (DESCR_SYS: Sys)
                                                   and (: TEXT = Text[22]));
      create DescrOf:= DESCR_OF((DO_DESCR: TechN)
                                                   and (DO_OBJ: Li));
    endif;
  else
    {
    Find the next POSITION of attributes in Eg
    }
    Pos:= 0;
    At:= first(ATTRIBUTE(AT_EL: Eg);
    while At <> () do
      if POSITION(: At) > Pos then
            Pos:= POSITION(: At);
      endif;
      At:= next(ATTRIBUTE(AT_EL: Eg);
    endwhile;
    Pos:= Pos + 1;
    {
    }
    Attr:= first(ATTRIBUTE(AT_EL: Es);
    if Attr = () then
      {
```

[20] DefaultRoleName(Ro, N1, N2) returns in N2 the value of N1 if N1 is a possible name for the role Ro of the SCHEMA; otherwise, N2 contains a 'correct' name proposed by the user.

[21] The value of *Text* is: 'The role ' + R_NAME(: Ro) + ' represents the specific entity type which is now represented by the generic entity type.'

[22] The value of *Text* is: 'The role ' + R_NAME(: Ro) + ' represents the specific entity type which is now represented by the generic entity type.'

```
        No attribute in Es: a new boolean attribute is created in Eg
        }
        create AttrC:= ATTRIBUTE((AT_EL: Eg)
                                        and (: AT_NAME = E_NAME(: Es))
                                        and (: FORMAT = boolean)
                                        and (: MIN_REP = 0)
                                        and (: MAX_REP = 1)
                                        and (: AVG_REP = AvgRep)
                                        and (: POSITION = Pos);
        addlist(ListAttr, AttrC);
        create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                                and (DESCR_SYS: Sys)
                                                 and (: TEXT = Text[23] ));
        create DescrOf:= DESCR_OF((DO_DESCR: TechN)
                                                 and (DO_OBJ: AttrC));
      else
        {
        The attributes of Es are grouped in a new attribute of Eg
        }
        create AttrC:= ATTRIBUTE((AT_EL: Eg)
                                        and (: AT_NAME = E_NAME(: Es))
                                        and (: FORMAT = compound)
                                        and (: MIN_REP = 0)
                                        and (: MAX_REP = 1)
                                        and (: AVG_REP = AvgRep)
                                        and (: POSITION = Pos);
        addlist(ListAttr, AttrC);
        create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                                and (DESCR_SYS: Sys)
                                                 and (: TEXT = Text[24] ));
        create DescrOf:= DESCR_OF((DO_DESCR: TechN)
                                                 and (DO_OBJ: AttrC));
        Pos:= 1;
        while Attr <> () do
          modify Attr((AT_EL: AttrC) and (: POSITION = Pos));
          Pos:= Pos + 1;
          Attr:= next(ATTRIBUTE(AT_EL: Es);
        endwhile
      endif;
    endif;
    {
    Put the description/technical note of each specific entity type in that
    of Eg
    }
    FindDescrTechN(Es, DescrEs, TechNEs);
    if DescrEs <> () then
      modify Descr(: TEXT = TEXT(: Descr) + TEXT(: DescrEs));
      delete DescrEs;
    endif;
    if TechNEs <> () then
      modify TechN(: TEXT = TEXT(: TechN) + TEXT(: TechNEs));
      delete TechNEs;
    endif;
    {
```

[23] The value of *Text* is: 'This attribute represents the specific entity type ' + AT_NAME(: AttrC) + ' which is now represented by the generic entity type.'

[24] The value of *Text* is: 'This attribute represents the specific entity type ' + AT_NAME(: AttrC) + ' which is now represented by the generic entity type.'

```
    Suppress Es and its 'characteristics' (the category is deleted if Es is
      the last specific entity type)
    }
    delete Es;
    {
    }
    Subt:= next(SUBTYPE(ST_GST: Cat);
  endwhile
  {
  Representation of the class constraints by a global cardinality if more
  than one specific entity type otherwise by a 'simple' cardinality
  constraint
  }
  if (sizelist(ListRole) + sizelist(ListAttr)) = 1 then
    Ar:= firstlist(ListRole);
    if Ar = () then
      Ar:= firstlist(ListAttr);
      if Cov then
          modify Ar(: MIN_REP = 1);
      endif;
    else
      if Cov then
          modify Ar((: MIN_CON = 1);
      endif;
    endif;
  else { At least two specific entity types }
    {
    Find the next available IKO_CODE of ID_KEY_ORD of Eg
    }
    Code:= 0;
    Iko:= first(ID_KEY_ORD(IKO_EL: Eg));
    while Iko <> () do
      if IKO_CODE(: Iko) > Code then
          Code:= IKO_CODE(: Iko);
      endif;
      Iko:= next(ID_KEY_ORD(IKO_EL: Eg));
    endwhile;
    Code:= Code + 1;
    {
    }
    if Cov  then
      if Disj then { The category formed a partition }
        create GlobCard:= ID_KEY_ORD((: IKO_CODE = Code)
                                              and (: TYPE = 'Glob_card')
                                          and (: PARAM1 = 1)
                                              and (: PARAM2 = 1)
                                              and (IKO_EL: Eg));
      else { The category formed a cover }
        create GlobCard:= ID_KEY_ORD((: IKO_CODE = Code)
                                              and (: TYPE = 'Glob_card')
                                          and (: PARAM1 = 1)
                                              and (: PARAM2 = 99999)
                                              and (IKO_EL: Eg));
     endif;
    else
      if Disj then { The category formed a disjunction }
        create GlobCard:= ID_KEY_ORD((: IKO_CODE = Code)
                                              and (: TYPE = 'Glob_card')
                                          and (: PARAM1 = 0)
```

```
                                                    and (: PARAM2 = 1)
                                                    and (IKO_EL: Eg));
      endif;
    endif;
    {
    Update of the components
    }
    Nr:= 1;
    Ro:= firstlist(ListRole);
    while Ro <> () then
      create Comp:= COMPONENT((C_IKO: GlobCard) and (C_RA: Ro)
                                                    and (: TYPE = 'Role')
                                                    and (: SEQ_NBR = Nr));
        Nr:= Nr + 1;
      Ro:= nextlist(ListRole);
    endwhile
    At:= firstlist(ListAttr);
    while At <> () then
      create Comp:= COMPONENT((C_IKO: GlobCard) and (C_RA: At)
                                                    and (: TYPE = 'Attr')
                                                    and (: SEQ_NBR = Nr));
        Nr:= Nr + 1;
      At:= nextlist(ListAttr);
    endwhile
  endif;
end;
```

Algorithm E.9

```
procedure CreateSpecificET (var E: record-var
                            var ListAttr,
                            ListRole: list-record-var);
{
Specification:

▪ Precondition:
                          • E does not have a category for which
                            class inclusion constraints are specified
▪ Effect:
                          • This algorithm implements the transformation
                            specified in 6.3.1.G.g (p. 90)
}
begin
  Sch:= SCHEMA(S_E: E);
  Sys:= SYSTEM(SYS_SCH: Sch);
  DefaultETName(E_NAME(: E) + '1', Name);
  DefaultETShortName(SHORT_NAME(:E) + '1', ShortName);
  create E1:= ENTITY_T ((E_S: Sch)
                                    and (: E_NAME = Name)
                                  and (: SHOR_TNAME = ShortName));
  Cat:= G_OF_SUBTYPE(GST_E: E);
  if Cat = () then { E1 will be the first specific ENTITY_T of E }
    create Cat:= G_OF_SUBTYPE((GST_E: E) and (: COVERING = false)
                                         and (: EXCLUSIVE = false));
  endif;
  create Subt:= SUBTYPE ((ST_E: E1) and (ST_GST: Cat));
  {
```

```
Transfer the optional attributes which stay optional in E1 (and also the
groups containing them)
}
Nr:= 1;
Att:= firstlist(ListAttr);
while Attr <> () do
  modify Attr((AT_EL: E1) and (: POSITION = Nr));
  Comp:= first((COMPONENT(C_RA: Attr);
  Code:= 1;
  while Comp <> () do
    Iko:= ID_KEY_ORD(IKO_C: Comp);
    modify Iko((IKO_EL: E1) and (: IKO_CODE = Code));
    Comp:= next((COMPONENT(C_RA: Attr);
    Code:= Code + 1;
  endwhile;
  Att:= nextlist(ListAttr);
  Nr:= Nr + 1;
endwhile
{
Transfer the optional roles which stay optional in E1 (and also the
groups containing them)
}
Ro:= firstlist(ListRole);
while Ro <> () do
  Ero:= E_ROLE(ER_R: Ro);
  modify Ero(ER_E: E1);
  Comp:= first((COMPONENT(C_RA: Ro);
  while Comp <> () do
    Iko:= ID_KEY_ORD(IKO_C: Comp);
    modify Iko(IKO_EL: E1);
    Comp:= next((COMPONENT(C_RA: Ro);
    Code:= Code + 1;
  endwhile;
  Ro:= nextlist(ListRole);
endwhile
{
Update the origin of E1
}
create Orig:= PROPERTY((PROP_O: E1)
                                  and (: P_ROLE = 'Origin')
                                   and (: P_VALUE = E_NAME(: E))
                                   and (: TYPE = 'ENTITY_T'));
{
Update the technical note of E1
}
create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                                  and (DESCR_SYS: Sys)
                                                   and (: TEXT = Text[25] ));
create DescrOf:= DESCR_OF((DO_DESCR: TechN)
                                                   and (DO_OBJ: E1));
end;
```

[25] The value of *Text* is: E_NAME(: E1) + ' contains the entities of ' + E_NAME(: E) + ' having eventually a value for the attributes/roles contained now in this entity type.'

Algorithm E.10

```
procedure DeleteSpecificET (var Eg, Es: record-var);
{
Specification:

▪ Precondition:
                          • ENTITY_T Eg has a category for which
                            class inclusion constraints are not
                            specified
                          • ENTITY_T Es belongs to that category and
                            has only optional attributes/roles
▪ Effect:
                          • This algorithm implements the transformation
                            specified in 6.3.1.G.h (p. 91)
}
begin
  Sch:= SCHEMA(S_E: E);
  Sys:= SYSTEM(SYS_SCH: Sch);
  Cat:= G_OF_SUBTYPE(GST_E: E);
  create Subt:= SUBTYPE ((ST_E: E1) and (ST_GST: Cat));
  {
  Transfer the optional attributes of Es which stay optional in Eg
  }
  {
  Find the next POSITION of attributes in Eg
  }
  Pos:= 0;
  At:= first(ATTRIBUTE(AT_EL: Eg);
  while At <> () do
    if POSITION(: At) > Pos then
      Pos:= POSITION(: At);
    endif;
    At:= next(ATTRIBUTE(AT_EL: Eg);
  endwhile;
  Pos:= Pos + 1;
  {
  Find the next available IKO_CODE of ID_KEY_ORD of Eg
  }
  Code:= 0;
  Iko:= first(ID_KEY_ORD(IKO_EL: Eg));
  while Iko <> () do
    if IKO_CODE(: Iko) > Code then
        Code:= IKO_CODE(: Iko);
    endif;
    Iko:= next(ID_KEY_ORD(IKO_EL: Eg));
  endwhile;
  Code:= Code + 1;
  {
  }
  Attr:= firstlist(ListAttr);
  while Attr <> () do
    DefaultAttName(Eg, AT_NAME(: Attr), Name)[26];
    modify Attr((AT_EL: E1) and (: AT_NAME = Name) and (: POSITION = Pos));
    Pos:= Pos + 1;
    Comp:= first((COMPONENT(C_RA: Attr);
```

[26] DefaultAttName(E, N1, N2) returns in N2 the value of N1 if N1 is a possible name of an attribute of ENTITY_T E; otherwise, N2 contains a 'correct' name proposed by the user.

```
    while Comp <> () do
      Iko:= ID_KEY_ORD(IKO_C: Comp) ;
      modify Iko((IKO_EL: E) and (IKO_CODE = Code));
      Code:= Code + 1;
      Comp:= next((COMPONENT(C_RA: Attr));
    endwhile;
    Att:= nextlist(ListAttr);
  endwhile
  {
  Transfer the optional roles which stay optional in E1
  }
  Ro:= fisrtlist(ListRole);
  while Ro <> () do
    Ero:= E_ROLE(ER_R: Ro);
    modify Ero(ER_E: E1);
    Comp:= first((COMPONENT(C_RA: Ro);
    while Comp <> () do
      Iko:= ID_KEY_ORD(IKO_C: Comp);
      modify Iko((IKO_EL: E1) and (IKO_CODE = Code));
      Code:= Code + 1;
      Comp:= next((COMPONENT(C_RA: Ro);
    endwhile;
    Ro:= nextlist(ListRole);
  endwhile
  {
  Update the technical note of Eg
  }
  if TechN <> () then
    modify TechN(: TEXT = TEXT(: TechN) + Text[27]);
  else
    create TechN:= DESCRIPTION((: TYPE = 'Tech_n')
                                          and (DESCR_SYS: Sys)
                                           and (: TEXT = Text[28]));
    create DescrOf:= DESCR_OF((DO_DESCR: TechN)
                                           and (DO_OBJ: Eg));
  {
  Delete Es (and the category if Es is its last SUBTYPE
  }
  delete Es;
end;
```

Algorithm E.11

```
procedure RepresIsaByRTs (var Eg: record-var);
{
Specification:
```

- Precondition:
  - Eg is an ENTITY_T with a category
  - Eg has only 'local' components in its groups
- Effect:
  - This algorithm implements the

[27] The value of *Text* is: 'The specific entity types have been represented by the generic entity type.'

[28] The value of *Text* is: 'The specific entity type ' + E_NAME(: Es) + ' has been suppressed in Eg.'

```
                        transformation specified in 6.3.1.G.i
                        (p. 91)
}
begin
  Sch:= SCHEMA(S_E: Eg);
  Sys:= SYSTEM(SYS_SCH: Sch);
  Cat:= G_OF_SUBTYPE(GST_E: Eg);
  Cov:= COVERING(: Cat);
  Disj:= EXCLUSIVE(: Cat);
  Subt:= first(SUBTYPE(ST_GST: Cat);
  newlist(ListSpec);
  newlist(ListRole);
  while Subt <> () do
    Es:= ENTITY_T(E_ST: Subt);
    create Ero1:= E_ROLE(ER_E: Es);
    create Ero2:= E_ROLE(ER_E: Eg);
    create Ro1:= ROLE((R_ER: Ero1)
                          and (: R_NAME = 'is-a')
                          and (: MIN_CON = 1)
                          and (: MAX_CON = 1)
                          and (: AVG_CON = 1));
    if POPULATION(: Es) <> NULL and POPULATION(: Eg) <> NULL
    then
      AvgCard:= POPULATION(: Es) / POPULATION(: Eg);
    else
      AvgCard:= NULL;
    endif
    create Ro2:= ROLE((R_ER: Ero2)
                          and (: R_NAME = 'may-be-a')
                          and (: MIN_CON = 0)
                          and (: MAX_CON = 1)
                          and (: AVG_CON = AvgCard));
    addlist(ListRole, Ro2);
    DefaultRTName(SHORT_NAME(Es) + '_ISA_' + SHORTNAME(: Eg), Name)[29];
    DefaultRTShortName(Name[1..7], ShortName)[30];
    create Li:= LINK((L_R: Ro1) and (L_R: Ro2) and (L_S: Sch)
                                and (: L_NAME = Name)
                                and (: SHORT_NAME = ShortName)
                                and (: DEGREE = 2)
                                and (: POPULATION = POPULATION(: Es));
    {
    Update the origin of Li
    }
    UpdateOriginLi;
    {
    Update the technical note of Li
    }
    UpdateTechNLi;
    {
    }
    addlist(ListSpec, Es);
    delete Subt;
    Subt:= next(SUBTYPE(ST_GST: Cat);
  endwhile
```

[29] DefaultRTName(N1, N2) returns in N2 the value of N1 if N1 is a possible name for a LINK of the SCHEMA; otherwise, N2 contains a 'correct' name proposed by the user.

[30] DefaultRTShortName(N1, N2) returns in N2 the value of N1 if N1 is a possible short name for a LINK of the SCHEMA; otherwise, N2 contains a 'correct' name proposed by the user.

```
{
Representation of the class constraints
}
if (sizelist(ListSpec) = 1 and Cover) then
  Ro:= firstlist(ListRole);
  modify Ro(: MIN_CON = 1);
else { At least two specific entity types }
  {
  Find the next available IKO_CODE of ID_KEY_ORD of Eg
  }
  Code:= 0;
  Iko:= first(ID_KEY_ORD(IKO_EL: Eg));
  while Iko <> () do
    if IKO_CODE(: Iko) > Code then
        Code:= IKO_CODE(: Iko);
    endif;
    Iko:= next(ID_KEY_ORD(IKO_EL: Eg));
  endwhile;
  Code:= Code + 1;
  {
  }
  if Cov  then
    if Disj then { The category formed a partition }
      create GlobCard:= ID_KEY_ORD((: IKO_CODE = Code)
                                          and (: TYPE = 'Glob_card')
                                    and (: PARAM1 = 1)
                                          and (: PARAM2 = 1)
                                          and (IKO_EL: Eg));
    else { The category formed a cover }
      create GlobCard:= ID_KEY_ORD((: IKO_CODE = Code)
                                          and (: TYPE = 'Glob_card')
                                    and (: PARAM1 = 1)
                                          and (: PARAM2 = 99999)
                                          and (IKO_EL: Eg));
   endif;
  else
    if Disj then { The category formed a disjunction }
      create GlobCard:= ID_KEY_ORD((: IKO_CODE = Code)
                                          and (: TYPE = 'Glob_card')
                                    and (: PARAM1 = 0)
                                          and (: PARAM2 = 1)
                                          and (IKO_EL: Eg));
    endif;
  endif;
  {
  Update of the components
  }
  Nr:= 1;
  Ro:= firstlist(ListRole);
  while Ro <> () then
    create Comp:= COMPONENT((C_IKO: GlobCard) and (C_RA: Ro)
                                          and (: TYPE = 'Role')
                                          and (: SEQ_NBR = Nr));
      Nr:= Nr + 1;
    Ro:= nextlist(ListRole);
  endwhile
 endif;
end;
```

'Toolbox'

Algorithm E.12

```
procedure StatInfer (var Eg: record-var);
{
Specification:

▪ Precondition:
                          • Eg is an ENTITY_T with a category forming
                            a partition
▪ Effect:
                          • The statistical inference rules (p. 78)
                            are applied on Eg
}
begin
  if POPULATION(: Eg) <> NULL then
    GenPop:= POPULATION(: Eg)
    NumberUnspecified:= 0;
  else
    GenPop:= -1;
    NumberUnspecified:= 1;
  endif;
  Cat:= G_OF_SUBTYPE(GST_E: Eg);
  Subt:= first(SUBTYPE(ST_GST: Cat));
  SpecPop:= 0;
  while (Subt <> ()) and (NumberUnspecified ≤ 1)  do
    Es:= ENTITY_T(E_ST: Subt);
    if POPULATION(: Es) <> NULL then
      SpecPop:= SpecPop + POPULATION(: Es);
    else
      Spec:= Es;
      NumberUnspecified:= NumberUnspecified + 1;
    endif;
    Subt:= next(SUBTYPE(ST_GST: Cat));
  endwhile;
  if NumberUnspecified = 1 then
    if GenPop <> -1 then
      modify Spec(: POPULATION = GenPop - SpecPop);
    else
      modify Eg(: POPULATION = SpecPop);
    endif;
  endif;
end;
```

Algorithm E.13

```
procedure FindDescrTechN (var Obj, Descr, TechN: record-var);
{
Specification:

▪ Precondition:
                          • Obj is an ATTRIBUTE, an ENTITY_T, a LINK
                            or a SCHEMA
▪ Postcondition:
                          • Descr is the DESCRIPTION with TYPE =
```

```
                          'Descr' associated with Obj, otherwise
                          Descr = ()
                        • TechN is the DESCRIPTION with TYPE =
                          'Tech_n' associated with Obj, otherwise
                          TechN = ()
}
begin
  Descr:= ();
  TechN:= ();
  DescrOf:= first(DESCR_OF(DO_OBJ: Obj));
  if DescrOf <> () then
    De:= DESCRIPTION(DESCR_DO: DescrOf);
    if TYPE(: De) = 'Descr' then
      Descr:= De;
    else { TYPE = 'Tech_n' }
      TechN:= De;
    endif;
    DescrOf:= next(DESCR_OF(DO_OBJ: Obj));
    if DescrOf <> () then
      De:= DESCRIPTION(DESCR_DO: DescrOf);
      if TYPE(: De) = 'Descr' then
        Descr:= De;
      else { TYPE = 'Tech_n' }
        TechN:= De;
      endif;
    endif;
  endif;
end;
```

Algorithm E.14

```
procedure CopyLink (var Li, Ro: record-var
                    var ListEnt: list-record-var);
{
Specification:

▪ Precondition:
                        • Li is a LINK and Ro is a ROLE attached to Li
                        • ListEnt contains ENTITY_Ts which are specific
                          ENTITY_Ts of the ENTITY_T playing the role Ro
▪ Effect:
                        • The role Ro is replaced by a multidomain role[31]
                          concerning the ENTITY_Ts of ListEnt
                        • Li is however not deleted
}
```

Algorithm E.15

```
procedure CopyAttr (var Att, Obj: record-var);
{
Specification:

▪ Precondition:
                        • Att is an ATTRIBUTE
                        • Obj is either an ATTRIBUTE, an ENTITY_T or a LINK
                        • POSITION(: Att) and AT_NAME(: Att) are supposed
                          acceptable for Obj
```

[31] Actually, it is replaced by the replacement of a multidomain role (see footnote 8, page 87).

```
▪ Effect:
                          • A copy of Att is created
                            and attached to Obj
}
begin
  create A:= ATTRIBUTE((AT_EL: Obj)
                       and (: AT_NAME = AT_NAME(: Att))
                       and (: FORMAT = FORMAT(: Att))
                       and (: LENGTH = LENGTH(: Att))
                       and (: DECIM = DECIM(: Att))
                       and (: MIN_REP = MIN_REP(: Att))
                       and (: MAX_REP = MAX_REP(: Att))
                       and (: AVG_REP = AVG_REP(: Att))
                       and (: AVG_LENGTH = AVG_LENGTH(: Att))
                       and (: POSITION = POSITION(: Att)));
  {
  Copy of the description/technical note
  }
  CopyAttObjDescrTechN;
  {
  Copy of the component attributes if any
  }
  AttComp:= first(ATTRIBUTE(EL_AT: Att));
  while AttComp <> () do
    CopyAttr(AttComp, A);
  endwhile
end;
```